Requirenautics Quarterly

The Newsletter of the Requirements Engineering
Specialist Group of the British Computer Society

http://www.resg.org.uk

RE-Locations

RE-Soundings

Editorial

Having to take a dose of your own medicine can be an enlightening experience. In a reversal of the proverbial poacher-turned-gamekeeper scenario, I've recently found myself on the receiving end of an RE process.

It's a multi-million pound bespoke project with a range of stakeholders including the user group that I represent. Requirements elicitation has involved a great deal of helping the designer team (who haven't worked in this domain before) understand our operations in order to be able to interpret our requirements. So far, we've been through a feasibility study, had focus groups and user group meetings and meetings to rank and prioritise our sometimes conflicting requirements.

It's reinforced for me a number of important points:

- How difficult it is for users to articulate what they do and why;

- How much we expect of users when we ask them to work out what their requirements are and decide on what they are and are not prepared to accept;

- How the organisational environment introduces political requirements that may override operational requirements in - I must be careful here - a way that is sometimes non-optimal.

No wonder systems are hard to get right. Except that in this case, it isn't a system - it's a building. Our design team are architects, structural engineers, quantity surveyors, etc., with the architects acting as the requirement engineers.

Ok. If you've read this far, I'd better correct my implied characterisation of the relationship between requirements engineer and user that will have got at least some of you shouting 'No!'. It's an inexact analogy, granted, but surely the user should be closer to gamekeeper than poacher. It's just that that isn't how it feels from my new perspective as one-of-many stakeholders.

This issue of RQ contains an innovation - a readers' letters section. In the famous words of Humphrey Littleton, we've been inundated with a letter (though not from a Mrs Trellis of North Wales). I'd welcome more in future. And articles are always welcome.

Pete Sawyer
Computing Department, Lancaster University.

Chairman's message

Turnout at the last few RESG meetings has been increasing. Perhaps interest in requirements engineering is on the rise again? Or perhaps we are organising more topical meetings? One thing is for certain, places at our 10th July meeting in London on Scenarios are already filling up, so I encourage you to pre-register - attendance is free!

I do have ulterior motives for inviting you to attend the July event. It will be preceded by the group's Annual General Meeting (AGM), and I would like to invite all RESG members to come along and hear what the group has been doing over the last year, and to contribute to the planning of future programme of activities.

You should have received by now the message from Steve Armstrong, RESG Membership Secretary, requesting that your renew' your membership (free for 2002!). If you have not already done so, please renew as a matter of urgency as Steve is in the process of updating and purging the RESG membership database, so this could be the last issue of RQ that you receive if your details have not been validated.

Looking forward to seeing many of you in July for the AGM and in September for the RE'02 conference in Germany.

Bashar Nuseibeh
The Open University

RE-Treats

Next event organised by the group

Scenarios Work! Improving Requirements Engineering with User Cases and Scenarios

Date: 1.30 - 5.15pm 10th July 2002
Location: Pearson 229, University College London, Gower Street, Liondon
Contact: Ian Alexander.
(iany@easynet.co.uk)

Scenarios and Use Cases are becoming seen as the natural way to organize requirements, especially when object-oriented software development is to follow. However, some researchers and practitioners believe scenarios are far more widely applicable in systems and conventional software. Others argue that Use Cases have been over-hyped and misapplied, whether to systems for which they are not ideal or using techniques that are far from best practice. This meeting debates the benefits and dangers of applying a range of Scenario and Use Case techniques in systems and software development.

This half-day event, composed of expert presentations from widely differing points of view, will investigate precisely how and when to document requirements using scenarios and use cases. There will be an opportunity for questions and discussion immediately following each speaker's presentation.

Requirements, Risk and Value in Systems Engineering

Organised by the EPSRC funded SIMP project.
Date: 10.30am - 2.45pm, 18th July 2002
Location: City University, London
Contact: Raquel Monja, Centre for HCI Design, City University. (r.monja@city.ac.uk)

This event, held half-way through the EPSRC-funded SIMP project, reports intermediate results as a important contribution to the UK's expertise in systems engineering and systems integration. The event is also intended to explore the exploitation of SIMP's results, and to build links with other systems engineering research institutions, projects, programmes and research-active organisations.

RE-Calls

Recent Calls for Papers and Participation

Requirements Engineering Challenges and Issues for Developing e-Government Electronic Public Service Delivery (EPSD) Software Systems

To be held in conjunction with the 6th World Multi Conference on Systemics, Cybernetics and Informatics, July 14 – 18, 2002, Orlando, Florida, USA.

http://homepages.feis.herts.ac.uk/~resg/html/upd._08_11_01_a.html

Most governments are currently engaged in ambitious, tight-scheduled e-government Electronic Public Service Delivery (EPSD) initiatives. The goal of these e-government initiatives is to provide the business

community and citizens with an efficient and effective public service. However, the e-service delivery challenges for e-government are profound. Citizens and business organizations that need to deal with the government should have a choice of channels for accessing government services. Private and voluntary sector organizations should be able to access both central and local government information in order to deliver their services to the citizens. The mobility of citizens over a wide geographical range raises new essential system and user requirements for the e-government EPSD system. To meet the challenges posed by these systems, new and efficient requirements engineering methods, processes, and techniques that are suitable for developing government-to-citizen (G2C), government-to-business (G2B) and government-to-government (G2G) systems are required. This session is seeking thought-provoking presentations and revealing case studies that break new ground in understanding the requirements engineering challenges and issues that will lead to practical steps in developing 'joined-up' e-government EPSD systems.

International Council on Systems Engineering 12th Annual Symposium (INCOSE 2002)

July 18 – August 1, 2002, Las Vegas, Nevada, USA.

http://www.incose.org/symp2002/

The Symposium theme "Engineering 21st Century Systems: Problem Solving through Structured Thinking" calls on delegates to discuss and debate the application of defined systems engineering processes to understand and solve the challenges that lie ahead in the definition and the design of new products and systems, the deployment of new technology, and the effective use and support of legacy systems that are still with us in this new century. The Symposium will provide a forum to address all aspects of 21 st Century problems and opportunities, the systems engineering principles, processes, methodologies and tools that can address them, and the resulting systems, products, and services.

Papers are invited that can make a contribution to the theme of the Symposium, within the framework of the traditional technical tracks and areas of special interest. All types of papers will be considered, including case studies, developmental work and technical analysis. Papers will be judged on a range of parameters including clarity of expression, effective communication of ideas, and technical content. Papers must be submitted in English, the official language of the INCOSE 2002 Symposium.

28th Euromicro Conference, Software Process and Product Improvement

September 4 - 6, 2002, Dortmund, Germany.

http://www.sea.uni-linz.ac.at/SPPI2002

In today's competitive world the quality of software systems is a key to economic success and stability. The Software Process and Product Improvement track concentrates on processes, methods, and tools improving the quality of software products.

Suggested topics of interest include, but are not restricted to:

- Software process assessment and improvement
- Organisational and business views to process improvement
- Quantitative models for development processes and products
- Distributed software development and virtual organisations
- Process and product improvement for e-business application engineering
- Use and usefulness of quality standards for software products and processes
- Approaches for modelling and enacting software processes
- Lightweight and flexible approaches
- Processes for component-based software development
- Verification and validation of software products
- Approaches improving dependability of software systems
- Industry best practice experiences and case studies in above areas

The framework of the track will contain sessions with a special focus. Proposals for special sessions or panels are welcome. Please send suggestions to the program chair. Special session titles will be posted on this web page.

IEEE Joint International Requirements Engineering Conference (ICRE'02 and RE'02)

September 9 – 13, 2002, University of Essen, Germany.

http://www.re02.org

RE '02 celebrates two major milestones in RE

a) the 10th Anniversary of international IEEE RE conferences and symposium

b) the foundation of a new conference series, the IEEE Intl. Requirements Engineering Conference (RE) which results from the "unification" of the former IEEE Intl. Conference on RE (ICRE) and the IEEE Intl. Symposium on RE (RE)

The conference will include a technical and an industrial paper track, with refereed papers describing novel research, industrial problem statements, experience reports and surveys. The program also includes keynote speakers, state-of-the-art and practice

tutorials, and an exhibition, which includes companies with RE tools and services, book publishers, and other related exhibitors. There will also be several associated workshops, including a doctoral workshop for PhD students, as well as research demos and tools.

The RE 02 conference will provide an opportunity for practitioners and researchers to share ideas and experiences, while enjoying the hospitality of the University of Essen.

RE-Readings

Reviews of recent Requirements Engineering events.
All reports by Ian F. Alexander

ISEN: Interdisciplinary Software Engineering Network

The Royal Society, St James', London, 11 December 2001.

Prof. **Paul Layzell** (UMIST) **welcomed a good crowd** of about 80 people to the splendour of the Royal Society for the inaugural ISEN meeting: he was evidently pleased and surprised at the turnout (which included 5 members of the RESG committee). The meeting was part of the arrangement with the EPSRC to promote engagement with the software community. The ISEN philosophy was about issues surrounding software, such as complexity, economics, the supply chain, the marketplace, privacy, dealing with system failures, and risk management.

Prof. **Bashar Nuseibeh** (OU) spoke on **A Roadmap of Software Systems Requirements Engineering**. He illustrated his theme with the 'probably untrue' tale of the space race pen for use in zero gravity. NASA spent millions. The Russians gave the cosmonauts pencils. He gave a simple and plain introduction to the engineering and economic arguments in favour of requirements, quoting Pamela Zave on the nature of Requirements Engineering.

Warming to his theme he explained why he preferred elicitation to capture, speaking about negotiation and adjusting one's expectations; "not 'shalls' but system boundaries", and "writing 'em down doesn't mean they're agreed". Out of this comes the whole body of work on validation, negotiation, conflict resolution and prioritisation, not to mention tolerating inconsistencies, his personal crusade. Impact analysis was crucial; adding or deleting requirements meant fixing errors, managing the inconsistency, and careful configuration management. Evolution was basic in all product families, such as for mobile phones; the goal was to deal with the more volatile requirements without affecting the system architecture.

Allen Fairbairn spoke in his relaxed and fluent manner on **How Intelligent Enterprises Handle Evolving Requirements**. He reminded us of the project manager's wisdom on horrors like Eurotunnel; the true phases of such a project are enthusiasm, disillusionment, blame for the innocent and reward for the uninvolved (namely, the lawyers). He was Systems Engineering manager on the Trans Manche Link: "We were almost as siloed as the rest of the organisation". It worked well, but all the knowledge gained was instantly dissipated and the lessons were lost, not learnt.

Requirements were not static. In the classical view, they were givens, fixed inputs to the fire-and-forget 'ballistic' style of project. But in the case of the tunnel, the competition – the ferry operators – bought faster and more comfortable boats, and now they take just 1 hour to cross the channel in stabilized luxury. "If the banks had known this they'd never have funded the tunnel". Every project changes the state of the world around it: $S_{after} \neq S_{before}$, which is a major challenge to all big projects. The project needs to consider not only its product, but other entities including processes, people, and infrastructure; actors including not just the customer but competitors, suppliers, and other stakeholders; and attributes including the market, legacy systems, competency, and culture. These must all be seen together. The result is a big picture of the extended enterprise, the supplier network or whatever you like to call it.

Prof. **Margaret Bruce** (UMIST) spoke about **A Framework for Requirements Capture**. Idea generation for new products was generally poorly defined, often happened by chance, and led to 46% of resources being devoted to product failures. Coming from the world of product design, she went over all the arguments familiar to requirements engineers for doing things better. Was she reinventing the wheel, I wonder? She suggested that one should analyse requirements to check the fit with the problem, to consider the cost and market impact, and whether the needed skills were available. She presented a simple flowchart running from idea generation (divergent), idea screening (convergent), stakeholder identification, requirement selection to discard loss-makers early, and some editing steps. This did sound more like a product management perspective on things in requirements that her team ought to have known about, rather than material for a new book (Wiley, 2001).

Martin Whitehead (Head of Information Security, Co-operative Bank) spoke on **Managing Evolving Requirements**. He spoke mostly about **Security Requirements**, with some asides about evolution not being especially significant in practice. Security was all about people, controls, and the nature of the security problem, he opined. "I'm a paranoid control freak, but I've only said 'No' one in 11 years". The basic

requirements were for '24x7', i.e. round-the-clock availability such as for a bank's website, with steadily rising bandwidth. Security meant staff, PINs and photographs or other means of identification, screening and training. With an internet-facing system it also meant new requirements and new code, extra customer scrutiny and control of CGI and other scripts to prevent loopholes. Success depended on testing with "contingency plans for overwhelming success". Otherwise you got your knuckles rapped by the regulator, as when Egg's credit card was so popular that their website couldn't cope.

We relaxed and chatted over a very pleasant stand-up lunch in the elegant foyer.

Prof. **Keith Bennett** (Durham) introduced the **Key Issues in Managing Evolving Requirements**, as a warm-up for the **Group Working Sessions** into which we then broke to spend the afternoon in a sort of brainstorming mode. Keith Bennett's was an excellent, thought-provoking, and all too brief introduction. Once upon a time the motto in business was "Big eats Small". Now it was "Fast eats Slow". The problem was that requirements evolved, so the IT needs of business had to evolve too. What was evolution: was it Darwinian or a random walk? Did you need to model emergent behaviour or feedback? How could projects manage evolution better? Solutions did not have to be technical; for instance, reuse of software is strongly to do with reward structures. For the groups, he asked us two questions: What inhibits software from evolving? And How can one enable software to grow faster?

The Group Working Sessions were fun, being enjoyably divergent with a fresh group of people, and work at least for the hard-pressed co-ordinators who had the job of trying to assemble something coherent from their groups. Our group of 8 people came up with a dozen mechanisms for each of the two questions. The co-ordinator of each group – ours was Sue Black (South Bank University) – then had to try to come up with just two key points in a five-minute presentation: an impossible task given the very successful divergence of the group work.

The **obstacles** summarized by the groups were: Inflexible architectures; inflated expectations; inattention to -ilities; interdependence of systems; legacy systems; conflicting interests; malignant data structures (don't ask); complexity; and inconsistency of worldview.

The **promising strategies** summarized by the groups were: Clever people; dialogue; communication; education; capturing assumptions and requirements; component-based architectures; decentralisation; autonomous components; and 'good enough' systems.

As you can see from these lists, there certainly isn't just one 'right' view of either the problem or the solution, though at least people seemed to agree there was a problem worth addressing. Obvious inconsistencies in the rough-and-ready selection include that silo-ism isn't among the obstacles, though dialogue and communication are among the solutions; while requirements management is evidently a key part of the solution, but inability to trace and assess impact of change isn't listed among the problems. Perhaps a fuller listing of the findings, coupled with a clustering instead of a sampling of topics would be better. Still, it was an excellent idea to have a participative session instead of wall-to-wall talks.

The sessions were followed up by the creation of ISEN mailing lists to support discussion, but at least in our group this petered out after a few days, as people simply gave position statements and then fell silent. Engagement is evidently not something you can conjure up in a few moments, even with the delicious canapés of the Royal Society. All of us who try to serve industry by supporting groups like the RESG, the IEE, and the admirable SRE mailing list know how hard it is to create and sustain dialogue and participation. ISEN is to be welcomed for its part. Let us hope that the dialogue steadily builds with subsequent meetings.

Key Challenges in Safety Requirements Engineering

Praxis, Bath, 17 February 2002.

Efi Raili welcomed everyone in a large but packed room to Praxis and Bath. It was certainly very pleasing to see so many people attending a meeting outside London.

Colin Brain (QinetiQ) spoke on Jackson Problem Frames in Safety Requirements Management. He confessed to some technological pessimism with respect to misuse of technology - in contrast to 18th century optimism about the Industrial Revolution. Most engineers nowadays did not like abstract thinking, he argued; instead, they like to reason about solutions. Michael Jackson's Problem Frames let engineers reason about problems, provoking discussion about and better understanding of problems, and then promoting reuse of solutions as they allowed multiple layers of a problem to be explored. For instance, a high-level user need might be satisfied at a lower level by the capability of a piece of equipment, along with the training, manpower, organisation, support and maintenance, and doctrine (standard procedures) that went with it. These in turn could be decomposed at a still lower level.

He had applied Problem Frames in quite a simple way to identify entities and their environments, linked to each other by requirements. For example, a vehicle was linked to its users by (naturally) Operational Requirements, whereas it was linked to a Synthetic Environment (a simulator) by Validation Requirements.

Bashar Nuseibeh said that this made use of just one type of Problem Frame, and that layering itself caused

difficulty, as it led to functional decomposition, not to mention complexities such as multiple safety cases.

Tim Kelly (University of York) spoke on 'From Requirements to Safety - and Back Again'. This was the same talk that he gave at RE'01 in Toronto (*see* RQ #25), as he had recently been ill; we look forward to hearing him talk about how Goal-Based Requirements Decomposition relates to Safety Cases on another occasion. Fortunately it was an excellent talk that merited a second hearing by the small percentage of the audience that had been in Toronto, and Tim was in quotable mood.

"People do a FHA (Functional Hazard Assessment) and say, 'Oops, what are the Functions?' – this isn't really acceptable, as they've then duplicated the functions."

Obviously, systems engineers must feed core requirements into the safety process: the streams are distinct but must never be independent. What is more, FHA is much better done with scenarios or use cases than a "naked list of functional requirements" as then you understand how they fit together. But "one thing I dislike about use cases is the randomness of Alternative Paths, when they can occur." In short, the very familiarity and cosiness of a text-based approach makes analysis more difficult. The use case approach also tends to focus attention on single-point failures – following a simple scenario leads you to ask 'what can happen here?' rather than thinking about all the simultaneous things that might be happening.

Could a safety analysis ever be complete? One could be complete with respect to a list of core (system) functions and guide-words: better than nothing.

Joanne Stoke (Praxis) very competently gave Alan Simpson's talk 'Will it Be Safe?' as he was unable to attend. She explained that Praxis' REVEAL method was an approach to engineering safety requirements based on Michael Jackson's work (again). The requirements in the World overlapped with the design of the Machine in the form of the interface or Specification of the Machine. From this you got Jackson's basic Satisfaction Argument, namely that

$$D.S \vdash R$$

(the symbol is pronounced 'turnstile'). This says that Domain knowledge combined with the Specification shows that the Requirements will be met.

The Safety version of this argument is that the risk in this particular Domain, such as aviation, combined with the Specification of this particular system, such as an aircraft, shows that the plane will be safe to fly in. To construct a proper Satisfaction Argument is a rigorous matter, involving fault trees, FHA (see above), other bottom-up analyses such as Functional Failure Analysis, and so on. You ended up with functional statements as well as safety properties. These last included logical statements of what should happen under what conditions; performance; tolerance; and assurance requirements such as the SIL (Safety Integrity Level).

REVEAL was proving itself to be a good and rigorous method for identifying safety requirements, offering a gain in clarity over earlier approaches.

Carl Sandom (Praxis) gave 'An Engineer's Perspective on Human Factors Considerations for Safety'. He said there was nothing new under the sun, illustrating this with a grim picture of the hull of the Herald of Free Enterprise, which sank after its owners had ignored requests for indicator lights.

Accidents were "10% technical, 90% human", he suggested. And of these human factors, cognitive ones were much harder than simple arguments about anthropometrics and whether people in the range 5th to 95th percentile could reach the controls. Incidentally, even if they could, 10% of the population would still be excluded!

Bad examples were easy to collect. He showed a pull-type handle on the outside of a door that in fact had to be pushed to open it. This was just bad Human Factors design, and it could become a serious hazard in a fire, causing or aggravating a panic as people struggled to escape.

Safety constraints were often unwelcome in a contract, as they meant extra cost. But going for the cheapest often sowed the seeds of failure; treating a system as a "bag of bits" didn't work.

Human Reliability Analysis (HRA) was difficult: how could you count 'error opportunities'? So Human Factors as a whole was a hard problem for safety analysis; human actions are complex and unquantifiable; it is hard even to set safety targets, and harder to demonstrate human integrity.

Recycling Requirements for Systems and Product Families

Imperial College, London, 14 April 2002.

Neil Maiden and **Alessandra Russo** welcomed us to this all-day meeting, consisting of a morning tutorial and a rich set of afternoon talks.

Prof. **Mike Mannion** (Glasgow Caledonian University) invited us to join in to his 3-hour tutorial and asked us in workshop style to introduce ourselves: we were a good mix of academics and industrialists, with domain practitioners (police, air traffic control) as well as consultants and software engineers, mostly British but with very welcome visitors from Calgary and Namur (not to mention speakers from Finland and Germany). We sat around a large table to participate.

Mike Mannion said that without product line tools and methods, you ended up starting from scratch every time, copying and pasting requirements, and getting into trouble with the same arguments over and over again. But current tools didn't seem to be quite ready for the task. The term 'Product Line' had numerous

meanings, clearly implying though a set of related products specified together in some way, and allowing for some form of reuse or recycling. But what did you do with new requirements? Products A and B might share many requirements; B and C might also do so; but gradually products A and D, E, and F might share very little. Juha Savolainen said that he thought all the requirements were more or less shared.

The product line ideas are 10 years old, and the best website is Carnegie-Mellon's SEI, which has a product line practice (PLP) initiative section.

Why was it so hard? Experts moved on; products were complex, and evolved; – a set of them even more so.

He'd worked at ESOC in Darmstadt on a reuse method to tie in with ESA's PSS-05-0 standards and procurement model. ESA writes traditional atomic requirements in English. But even standardized documents differed widely! A plan, a daily plan, a medium-term plan and even a long-term plan all meant the same thing but came from different mission-control specifications.

The idea was to use a 'lattice', with several structured hierarchies of requirements including 'discriminants' and covering several viewpoints. A discriminant is a Feature that makes one system different in the product line. Discriminants could be Mutually Exclusive, Lists of Alternative Options not mutually exclusive where at least one is chosen, and Single Optional Features – or combinations of these three. These aren't logically distinct but are practically useful. Juha Savolainen and Bashar Nuseibeh discussed whether product features were appropriate instead of talking about user needs. Ronald Stamper said that he understood requirements to be what people wanted to be able to do, not technical features of products, which was totally different and much more stable: not how a phone worked but how people could conduct a speech act. Bashar said that we could be pragmatic and accept there was a relationship between problems and solutions, between requirements and design features.

Variable requirements were tricky to handle; in theory you could parameterize requirements with e.g. "respond to @X commands" (global parameter) or "within $Y seconds" (local parameter) but it wasn't clear how this would work in practice.

Engineers love the freedom to make decisions. Copying and pasting to make a new set of requirements is fun! But it is obviously problematic; could it be managed in some way? Choices made freely could easily be illegal. Maybe you could validate the requirements chosen by checking some Boolean logic rules. A dependency is an AND; a mutually-exclusive discriminant is an XOR; a multiple discriminant is an OR, and so on. The result is that the engineer can be asked to reconsider the requirement model; unfortunately it doesn't show where the problem is. It would be better to use a lattice model to drive the selection (rather than allowing freedom to make mistakes and then try to fix them) however

uncomfortable that might be for freedom-loving engineers.

So, you start at the root of the tree. When you come to a multiple adaptor you just get to see all the choices as they are all available to you. If you choose one or more, you get their children in the hierarchy, and so on. When you come to a mutually exclusive choice you obviously just choose one and you get its children. Where there are no decision points in a subtree you get the requirements automatically. The result is a lot easier than free selection.

But sometimes a chosen parameter for one requirement might affect other requirements. Presumably you need cross-links of some kind. Ljerka Beus-Dokic said that in that case you had to make the parameter a requirement in its own right. Neil Maiden said he'd used the ISTAR approach from Toronto to model dependencies quite formally, allowing for model-checking. Ian Alexander said that maybe you could go down a tree of functions, but for availability and safety and other non-functionals, you needed to consider how they affected all the other requirements or you'd end with a system you couldn't build at all. Mike Mannion agreed saying that engineers wanted a breadth-first approach, visiting different parts of the tree in parallel, not diving depth-first down the tree and maybe getting stuck. Juha said that if you dictated the order of traversal you also dictated the architecture; dependencies were the issue. Ljerka said that if there were only a few discriminants you only needed to consider dependencies between them. Neil said that for usability requirements, you had a repeatable pattern of dependencies, e.g. increased usability generally means decreased security; maybe you could identify universal dependencies. Juha thought not; you had to think in terms of the architectures you might use, and he made a purely local dependency model for this purpose. Mike said that the method of selecting requirements was an open issue for product-line engineering.

PITO (the police IT organisation) used Use Cases to model business processes, which then mapped directly to system Use Cases; they formed goal hierarchies and clearly reduced the complexity of dependencies as each Use Case had its own preconditions expressed in problem-domain terms.

Industry found reuse difficult (said Diane Martini of Travelex): which business area would pay for it? The first project has no reason for spending extra effort on helping possible future projects! You could have a programme management board which looked at sharing across projects, for instance, she said. In Travelex there was a team of analysts allocated randomly to projects, but you could also allocate them to business areas. Free allocation meant analysts got wider experience but didn't really help with reusing requirements. Ljerka said that you could always reuse by reverse engineering 'on the side'. Diane said that standards and tools were important: they hoped that RUP would help. Ian said it was no good having a reuse department that lived in a cupboard away from

real projects – it was never taken seriously. Mike said that Mark Griss had seen at HP a rotation of good engineers into the product-line department for a while before going back into real projects to remember what using such things was like.

Mike Mannion said that ESOC wanted total control, so projects would not be allowed to add discriminant requirements. Ljerka said that in her domain you could perhaps have a core (like a software object library) and discriminants local to projects / products, but you couldn't legislate for all future projects. Ian said (Swartout and Balzer 1982) that specification and design were inextricably intertwined. Mike said he found it "heretical to have more than one product-line model and was quite evangelical about it". The religious language shows how passionately people hold views about this topic.

In conclusion, Mike said that product-line engineering was a step up from specifying single products; you needed to model the product line in accordance with a product line life-cycle, and develop single product models from that. This needed to be supported by specialized product-line tools which were not yet commercially available.

Tim Trew (Philips Research) spoke on Building Product Populations with Software Components. Philips makes TVs. In 1965 they contained no software; in 1990, 64 kBytes; in 2000 there is 2 MBytes (of which most people use 1%). There is wide diversity of price from portables to widescreen and projection TVs, analog and digital, different broadcasting standards, teletext, integral DVDs and hard disks, user interfaces with graphics and 3D animations, program guides and multimedia home platforms. Products can have almost any combination on these axes. Low-end products are now moving to the same architecture as high-end ones, but with simpler functions: a severe challenge for requirement and design reuse.

So there are 2 problems: software is getting bigger demanding more time and effort; but the market demands cheaper products, faster. There is a severe squeeze between these pressures. Philips thinks you need to move from decomposition of systems to a composition paradigm where products in a family are all composed of the same set of entities. This is hard. Philips' Koala model provides interfaces, parameterization, 3rd party binding, and gluing of components together. The result is a diagram of software components that looks much like an integrated circuit diagram – but the boxes aren't chips but blocks of software, and the interfaces aren't wires but C code. Unfortunately control software is hard to decompose, as the different functions are tightly coupled. So, there has to be horizontal communication (in the manner of IBM's pipes & filters architecture) between components as well as up and down the hierarchy. A common architecture is essential. Products and subsystem releases have to be carefully planned.

Requirement reuse is based on use cases. They compose elementary or composite functions (i.e. assemblages of other functions) sequentially, which in turn come from a requirements object model. Crucially, those functions map onto system components which form a layered architecture. The use cases contribute to requirements for the user interface state and control, which also calls on graphics and control devices. In addition, functions are used to provide autonomous behaviour (independent of the user interface). Currently over a hundred software engineers are using the approach.

Friedemann Kiedaisch (University of Ulm) spoke on Requirements Recycling for embedded systems in cars at DaimlerChrysler. As in Philips there are increasing numbers of software functions, but in addition these are often safety-related. In Telematics, the system has always been mainly software. DC specifies such systems and these specifications are interpreted by suppliers. The specification engineers are not computer scientists but engineers who would rather build systems than describe them, with predictable consequences.

This creates many challenges. Documents are often 500 pages long, rich in implementation detail, containing unsystematic cut-and-pasted material with no history or explicit dependencies. These documents are prone to error, with inconsistencies, over-specification, and gaps. Authorship is often forgotten. Clearly this is unsatisfactory in any complex system, let alone one that is safety-related.

The proposed approach is to create reusable Use Cases by reverse-engineering from the existing specifications, linked in a requirements tool to system requirements, which in turn will trace to product designs. Requirements will become more modular and hence more reusable. Reuse will be by browsing starting from the use case model as the root of a hierarchy. With Ian Alexander, we built some example use case models for car subsystems and built complete functional traceability to existing specifications in a DOORS database. The approach could clearly handle horizontal dependencies between subtrees as links, but this has not so far been attempted. Jeremy Dick pointed out that as in software, it is a goal to maximise Coherence and minimise Coupling between units. Ljerka Beus-Dokic said it sounded as if a hypertext would help; Ian replied that the use case model indeed generated a rich network, and this could be exported to HTML as a page per use case, with links to represent inclusions, exceptions, and actors.

Juha Savolainen spoke on Requirements Engineering for Product Lines at Nokia. Everyone agrees we should do this; the question is how. The usual claims are faster, better, cheaper. Requirements are often the only thing you can reuse, but good reusable specifications should make design and implementation easier to reuse too.

By definition, members of a product family share requirements. There is often a large set of similar requirements. Some vary and must as Mike Mannion said be handled specially. The task is far easier if requirements are written with reuse in mind.

Development of mobile phones is commonly driven by features, raising many problems. Features become more and more numerous; they creep; they interact; they have different types of dependency; they often reflect design and implementation decisions. Feature interaction is a critical issue.

Product Line (PL) is a market segmentation, risk management, resource sharing and brand recognition technique. Making a family of products similar thus brings many benefits, not purely technical or to do with cutting development costs (as other speakers tended to imply). As in Philips, the issues are multi-dimensional.

For instance, Nokia phones actually offer 4 different user interfaces – some have a big central context-sensitive button, others do not. Reuse can be by creating a lead product that creates assets that follow-up products can reuse. Or you can develop assets off-line and try to push them into products – a more risky and slower strategy, and hence rarely used in dynamic markets like mobile phones (though it's common in aerospace, for instance).

Product line development is dualistic – Juha illustrated this with the chinese yin/yang monad ☯ – consisting of Product requirements and Product-Line requirements. Where there are mandatory standards in a domain, it is easy to reuse those as requirements. In mobile phones the situation is too fluid for that, so the best approach seems to be to group products in some dimension(s) to obtain small domains that can share many requirements, such as for GSM 900 (with a mandatory published standard). Several Nokia phones (models 3310, 6110, 8210, 8890) share GSM 900; all but the 6110 also share GSM 1800.

Design decisions also determine what can be shared. Deductive properties can say whether all variants share a requirement, or are allowed to share it, or if a requirement is unshared. Declarative properties state that requirements are mandatory, multiple, single/exclusive, or optional (see Mike Mannion on this). Inconsistency is a positive thing: it may be actually wanted or accidental.

Reuse is here to stay. Selecting product groups is not only sensible but critical to keep reuse simple. Separating out product line and unshared requirements helps by keeping complexity and costs down.

Jeremy Dick (Telelogic) spoke on Requirements Management for Product Families, a Three Dimensional Approach that he and Ken Jackson developed. Being in a consultancy company within Requirements Management allows him to compare different domains and reflect on product family behaviour.

The dimensions of the framework are product evolution, composition, and traceability. Products not only evolve by themselves; they also cross-fertilise and evolve along with other products. Product versions replace each other; variants branch off, but there is little distinction between these as in practice they all live together. Subsystems also have their own versions and variants which may be quite unlike their product equivalents; so do components. The relationships between variants need to be managed.

The layers also need to fit together to form coherent configurations. This CM problem leads to the composition dimension.

Thirdly, there needs to be traceability between requirements and design elements for products and their lower-level components. The configuration says that this product is built from those components; the traces say how that is, in more detail.

How can these 3 dimensions be managed? For evolution, requirements are specialised (after copying) for new versions. Composition is essentially via traceability matrices; you get a cascade as a cell in this matrix becomes a row in the next matrix down, each subsystem then mapping onto several components, and so on.

Reuse is then built-in: a new version inherits all the merits and mistakes of the old one, with all its traceability. (But what it perhaps doesn't do is identify genuine product-line requirements as a discrete set.) Then you can add, remove, or upgrade components within the new version. As Mike Mannion said, you can describe product-line requirements by stating whether (discriminant) requirements are exclusive or non-exclusive choices. This can be implemented as 'rich traceability' using links with AND, OR, and XOR between parent and child functions. What is more, if a user need dictates one particular XORed choice, then the other XORed items in that set are in fact definitely excluded, even if other user needs might want to select them (this shows that each user need leads to a separate tree of traces, but that trees can overlap). The rich traceability captures a lot of the justification or rationale for choices made during projects as Satisfaction Arguments; rationale is otherwise often lost when key people leave. In theory there could be a large payoff from this. Alessandra Russo said the approach resembled Axel van Lamsweerde's KAOS method.

Neil Maiden quickly and expertly summarized the emerging themes for the panel discussion with the following 'emerging questions to think about':

- What types of requirements to represent, and how?

- Goal hierarchies, requirements dependencies, choice feedback

- Requirement- versus design-based choices?

- Where are the boundaries - "intertwining personified"?

- Consider requirement-architecture trade-offs?

- How to manage and fund product line engineering?

- Degrees of freedom of stakeholders and requirements?

- What types of new methods and software tools do we need - viewpoints, or variations on use cases?

- What levels of reuse - organisational requirements?

- Product lines, COTS software packages and software components - all part of the same problem?

The meeting adjourned to a nearby tavern for continuing discussions.

RE-Papers

Functional Requirements Specifications for Business Systems – A Test Analyst's View

M.J. Lawrence
Independent Contractor

I thought it might be of interest if a Test Analyst's perspective on RE was presented. I work mainly in acceptance testing for large enterprises whose business systems have developed organically over as long as forty years. Such enterprises normally have several quasi-autonomous legacy software systems that have been mercilessly hacked over time, and to which major changes are being made.

The bulk of the effort in acceptance testing of business systems is functional testing, and this can only be as 'good' as the input to acceptance test analysis. I wish in this article to draw attention to some issues with the format of functional RS's from the Test Analyst's viewpoint, especially where Use-Cases are involved.

As an Acceptance Test Analyst, my job is :

- To verify that any Requirements documents are unambiguous and correct;

- To uncover as many omissions as possible in any Requirements documents;

- To identify a set of test cases that will be 'sufficient' and 'efficient' to demonstrate that the delivered system is fit for operational use - my prime objective.

I am, therefore, a major 'customer' of Requirements Specifications (RS), and when the RS's are missing or inadequate, I know I have to devote a great deal of time and effort in 're-discovering' the requirements in order to have an adequate input to acceptance test analysis. Usually I have only days to analyse and document domains which have evolved over decades, therefore I have had to develop analytical methods which are both fast and accurate.

The most common format of functional RS with which I am presented is what is known as the 'Wuthering Heights' style. They consist of hundreds of pages of dense narrative, with perhaps only one concrete fact per page. They are usually written by people with little or no analytical knowledge and get abandoned at about second draft, as no-one can be bothered to read them again.

Another popular format is the 'Fragmentary' style. This has two forms: one being a document consisting of huge tables containing short statements of functional requirement, the other form being a database containing the same information. At first sight, the fragmentary format looks good: every requirement is uniquely identified and controlled, and is cross-linked to other requirements. Although tables are a great way for looking things up, they are a very poor way to *present* information, as people do not read them, they just skim over them. I have found that databases of fragments are felt to be inaccessible. However, my main objection to the fragmentary style of functional RS's is that business systems are not used in a fragmentary way, and so I have to put the fragments back together again to try to work out what all the end-to-end transaction-flows are, and this brings me to Use-Cases….

Use-Cases for business systems in the forms given in Schneider & Winters [1] and Cockburn [2] are no good to a Test Analyst; they are not 'test-ready'. Obviously, if the Use-Cases are untestable or permit only superficial test analysis, then this bodes ill for the programme as a whole.

Is there a better way ? Well, in my opinion, yes; from a Test Analyst's perspective, a much better way is to define the Use-Case in some form of Structured English in terms of a static data-oriented conceptual model. All the possible transaction flows can be seen, and decisions taken by either User or system are shown in-place.

The key input to producing the definition is the static conceptual model. I have found in practice that a UML class diagram works best, as it can show the large amount of detailed, precise information that is essential for the Test Analyst. UML has very rich semantics compared with earlier ERD-based methods, and UML class diagrams support the principle of multiple classification, which is extremely useful in test analysis. The Use-Case definition is then written in terms of the types, sub-types, states, cardinalities, existence/non-existence of objects and links, attribute boundary values, etc. from the conceptual model, and defines the basic Create/Amend/Delete/Enquire/Reorganize operations

which must be carried out during the transaction-flow. Business rules and events are included, and are shown explicitly at their point of application in the transaction-flow, i.e. in their context.

This form of Use-Case definition is complete in itself, and can form the basis of evolutionary delivery of fully-tested business functionality in an OO component-based development, provided the Open/Closed Principle is rigorously applied. Test analysis becomes simple : a Transaction-Flow Graph can be drawn straight from the Use-Case definition and test cases chosen. As the definition is in Structured English, it is eminently amenable to formal walkthrough and inspection.

Now, you may think that this sounds very like the old McMenamin & Palmer Event/Response analysis approach, but there is a major difference. The M&P approach broke up the transaction into separate processes beginning and ending on *terminators*, where a terminator was either a domain external to the system context or a datastore. My Use-Case definition ends when all possible operations resulting from the Use-Case have been defined, e.g. if we have a Use-Case 'Person wants Financial Advice', the longest transaction-flow would be to go through Adviser appointment, proposal, underwriting, medical underwriting, contract generation and finally commission payment. This goes a long way towards spotting strange combinations of sub-types, states, events, etc. which are business exceptions requiring special treatment or which must be unreachable. Other approaches can end up omitting them, with possibly deadly consequences later[1].

A major objection is that this form of definition is very 'program-like', and therefore inaccessible to the Workers and Users resulting in resistance. In practice, and I admit initially somewhat to my surprise, I found the very opposite to be true. The static model is defined in purely business domain terms, so every class on the static model is a concept completely familiar to the Workers. The Use-Case definition mirrors what actually happens when a Worker carries out the transaction for real, i.e. it reads back almost like a training manual. If necessary, the formality of the Structured English used can be varied depending upon the audience from very 'friendly' to PDL-like.

This approach produced an added benefit : in my experience, the Workers know their own jobs very, very well, and they also know their expectations of any system which is going to help them get their job done. The Workers involved in creating and reviewing the Use-Case definitions and the conceptual model have probably never seen the entire picture before, and soon they begin to come up with worthwhile improvements, both to the system and the process as a whole. This helps prevent later scope-creep.

I have not forgotten about the time-oriented view; if a business object has a non-trivial state machine, I supplement the models with Statecharts for those objects. Additional suites of test cases are then designed which take the objects through every possible valid lifecycle with further test cases which demonstrate that the objects cannot have any invalid lifecycles.

What about use-case diagrams and activity diagrams ? I think use-case diagrams are a waste of time at this level; they tell me nothing useful and can lead to a premature fragmentation of Use-Cases. Such decomposition is the business of the design phase. I find drawing activity diagrams 'cold' ineffective, but drawing an activity diagram from a Use-Case definition is a very powerful way of verifying the definition. It shows up 'dangling else's' and is particularly good at uncovering any missing communication objects which must pass across 'swimlanes'[2].

In conclusion, experience with this approach over the last six years has shown it to be fast and accurate. I have even been able to establish stable baselines from chaos. It results in a relatively small number of small documents, which are easy to maintain and configuration control. I am not claiming that this approach is any kind of 'silver bullet', but I do recommend it to the RE and Acceptance Test communities as 'a horse for a course'.

[1]   G. Schneider and J. Winters: 'Applying Use-Cases', Addison Wesley, 2001.

[2]   A. Cockburn: 'Writing Effective Use-Cases', Addison Wesley, 2000.

---

[1] In one domain, there were eight legacy systems which were being unified by putting a new modern application between the legacy systems and all Users. I was re-discovering the Requirements using this approach, and a business exception turned up. The two business consultants involved looked at each other, emitted several expletives, and immediately ran off to check one of the legacy system's data. They found and corrected a serious mistake which would have led both the organization and the Customers affected being prosecuted !

[2] Having uncovered a missing communication object, I set off to find out what was currently being used. It turned out that a core business transaction depended

entirely on either a Post-It stuck to a monitor or a scribbled note left on a desk. Needless to say, they

were often lost or ignored for days, thus delaying the transaction-flow.

RE-Sponses

I enjoyed reading Ian Alexander's contributions to RQ25. Concerning the Tools Digest from RE'01, caveat emptor indeed! Very pertinent questions - pity about the responses.

In the spirit of The Metaphors We Live By (referenced by Colin Potts in your review of RE'01), it seems to me

that another metaphor we are continually up against is "technology = solution". (The material in "Cunning Plans" by Hartswood et al. about misreading the nature of plans provides another example of this phenomenon.)

Gary Birch

RE-Sources

For a full listing of books, mailing lists, web pages and tools that have appeared in this section in previous newsletters, see the RQ archive at the RESG website: http://www.resg.org.uk

The requirement management place http://www.rmplace.org

A good general resource for RE issues. Includes Alan Davis' Requirements Bibliography.

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

An interesting collection of 72 papers (!) and a description of an ESPRIT project on co-operative requirements engineering with scenarios.

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

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

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

Reduced rates are available to all RESG members when subscribing to the REJ.

Mailing lists

RE-online (formerly SRE): http://www-staff.it.uts.edu.au/~didar/RE-online.html

The RE-online mailing list aims to act as a forum for exchange of ideas among the requirements engineering researchers and practitioners. To subscribe to RE-online

mailing list, send e-mail to majordomo@it.uts.edu.au with the following as the first and only line in the body of the message:

subscribe RE-online <your email address>

LINKAlert: http://link.springer.de/alert

A free mailing service for the table of contents of the *International Journal on Software Tools for Technology Transfer.*

RE-Bites

Ian Alexander provides this handy cut-out-and-keep acrostic for use cases:

Usage Scenarios, written as *<Actor> does <Action/Event>*

Stakeholders, and their Viewpoints, written as *<Role> wants <Desired Outcome>*

Exceptions and Alternative Paths, written as Usage Scenarios (with start events)

Conditions, including Trigger, required for Use Case to start

Actors, i.e. Roles, whether Human, System, or External

Success and Failure Guarantees, required to be true when Use Case ends

Extra details you consider necessary, like Local Constraints.

RE-Actors

The committee of RESG

Patron: Prof. Michael Jackson, Independent Consultant. E-Mail: jackson@acm.org.

Chair: Prof. Bashar Nuseibeh, Computing Department, Faculty of Maths and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-Mail: B.A.Nuseibeh@open.ac.uk.

Vice-Chair: Dr Alessandra Russo, Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK. E-Mail: ar3@doc.ic.ac.uk

Treasurer: Dr. Neil Maiden, Centre for HCI Design, City University, Northampton Square, London EC1V OHB, UK. E-Mail: N.A.M.Maiden@city.ac.uk.

Secretary: Dr. Wolfgang Emmerich, University College London, Department of Computer Science, Gower Street, London WC1E 6BT, UK. E-Mail: W.Emmerich@cs.ucl.ac.uk.

Membership secretary: Steve Armstrong, Computing Department, Faculty of Maths and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-Mail: S.Armstrong@open.ac.uk.

Newsletter editor: Dr Peter Sawyer, Lancaster University, Computing Department, Lancaster, LA1 4YR, UK. E-Mail: sawyer@comp.lancs.ac.uk.

Newsletter reporter: Ian Alexander, 17A Rothschild Road, Chiswick, London W4 5HS. E-Mail: iany@easynet.co.uk.

Publicity & WWW officer: Juan Ramil, Computing Department, Faculty of Maths and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-Mail: J.F.Ramil@open.ac.uk.

Regional officer & Chair for the North of England: Dr Kathy Maitland, University of Central England, Perry Bar Campus, Birmingham, B42 2SU. E-Mail: Kathleen.Maitland@uce.ac.uk.

Industrial liaison officer: Dr Efi Raili, Praxis Critical Systems, 20 Manvers Street, Bath BA1 1PX. E-Mail: efi@praxis-cs.co.uk.

Associate Industrial liaison officer: David Bush, National Air Traffic Services, UK. E-Mail: David.Bush@nats.co.uk.

Members-at-large:

Suzanne Robertson, Atlantic Systems Guild Ltd. 11 St. Mary's Terrace, London W2 1SU, E-Mail: suzanne@systemsguild.com