Requirenautics Quarterly

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

http://www.resg.org.uk

©2001, BCS RESG Issue 23 (March 2001)

RE-Locations

RE-Soundings

Editorial

Welcome to RQ23. One of the RESG's functions is to organise regular events for the RE community, and the last period has seen a number of these. Didn't make them? Then don't worry because the indefatigable Ian Alexander goes to them all for you. Read his reviews of the most recent in the RE-Readings section.

In addition to 'live' events, a number of recent RE-related books have appeared on the bookshelves. Why read them, when Ian Alexander will read them for you (RE-Publications)?

This issue isn't all reviews, however. It includes a number of articles, including a swansong by our very own Barbara Farbey. Sadly, this is Barbara's last contribution as an RESG committee member as she hands over the duties of Industrial Liaison officer to Efi Raili.

Another milestone is marked by the last of the regular series of Geoff Mullery's *CORE-Blimeys*. For many people, myself included, CORE-Blimey has been the highlight of RQ – a regular, consistent source of new ideas and insights. Geoff feels that he deserves a rest from RQ, but I hope we'll be able to squeeze occasional CORE-Blimeys from him in the future.

In the meantime, there is a vacancy for a regular RQ contributor. All applicants welcome! As always, RQ only exists through the efforts of the people who write the material (my own contribution is merely to glue them together). A huge proportion of this and most other issues of RQ is due to Ian Alexander. But we are always excited to receive contributions from the community at large – reviews, papers, experiences, thoughts and opinions. Indeed, RQ is really only viable if the community uses it to communicate their ideas. So please, if you'd like to contribute a piece, simply send it to me or, if you prefer, talk to me about what you'd like to contribute.

Pete Sawyer
Computing Department, Lancaster University.

Chairman's message

The Chinese have a curse: "May you live in interesting times". Well, the last few months have certainly been interesting, and the next few promise to be no less so.

Here at the RESG we haven't felt the curse bite yet, so we're carrying on regardless.

The RESG has continued to organise a varied programme of events, and our tireless reporter Ian Alexander continues to provide great accounts of what happened at those events to those RQ readers who couldn't make it to the meetings. And here I pause for a digression and a plea: we're still putting together events on topics that we, the RESG committee, feel may be of interest to you, the members. We'd much prefer if you told us the topics you'd like covered at these meetings. Even better, have you considered presenting at one of our meetings? Or even - gasp! - helping us organise one? Please do get in touch and talk to us - we're in the book.

2001 looks like having an international flavour for RE aficionados. You can enjoy Requirements and Architectures at the ICSE-2001 STRAW workshop in Toronto (Canada) in May; Requirements and Quality at REFSQ'01 in Interlaken(Switzerland) in June; and Requirements, Requirements, Requirements at RE'01 in Toronto (yes, Canada again) in August. But if don't fancy travelling, why not just sign up to attend our own Mini-tutorial on Requirements Negotiation in London on 5th April. The renowned Professor Barry Boehm, from the University of Southern California, will be presenting the half day afternoon tutorial to a limited audience of 30 people. So book now to avoid disappointment!

On a personal note, you may have noticed that my affiliation has changed. At the beginning of January I started a new job at the Open University (OU) as Chair for Computing. I will be continuing my own RE research work in the Computing Department at the OU, working with existing groups in the areas of Software Engineering, Human-Computer Interaction, and Computing Education. I am fortunate to be joined at the OU by RESG Patron, Michael Jackson, who has accepted a position as a Visiting Research Professor. Fancy an RESG meeting in Milton Keynes?!

Finally, David Shearer and Martina Doolan, from the RESG Membership Department, will by now have sent you a request to renew your RESG membership. If you have not already done so, please do complete and return your renewal as soon as possible, so that you continue to receive your copy of the RQ newsletter and discounts when attending RESG events. I hope you still feel that the RESG is good value for money and that it is worth the nominal subscription fee.

To new members: welcome!

Thank you and goodbye, Barbara

The RESG Industrial Liaison Officer, Dr. Barbara Farbey, is retiring, and is therefore also retiring from the RESG Executive Committee. Barbara has been the enthusiastic voice of industrial liaison for the RESG for about two years, and her skills and expertise will be sorely missed. On behalf of the RESG, RQ would like to wish Barbara all the best for the future.

Efi Raili, from Praxis Critical Systems, will be joining the RESG Executive Committee as the new Industrial Liaison Officer.

Bashar Nuseibeh
The Open University

RE-Treats

Next event organised by the group

Using Win-Win for Requirements Negotiation – a mini tutorial

Date: 5th April
Location: Department of Computing, Imperial College, London
Contact: Ms. Raquel Monja, City University. (R.Monja@city.ac.uk)

Speaker:
Barry W. Boehm is TRW Professor of Software Engineering and Director, Center for Software Engineering, University of Southern California. Barry Boehm received his B.A. degree from Harvard in 1957, and his M.S. and Ph.D. degrees from UCLA in 1961 and 1964, all in Mathematics. Between 1989 and 1992, he served within the U.S. Department of Defense (DoD) as Director of the DARPA Information Science and Technology Office, and as Director of the DDR&E Software and Computer Technology Office. He worked at TRW from 1973 to 1989, culminating as Chief Scientist of the Defense Systems Group, and at the Rand Corporation from 1959 to 1973, culminating as Head of the Information Sciences Department. He was a Programmer-Analyst at General Dynamics between 1955 and 1959. His current research interests focus on integrating a software system's process models, product models, property models, and success models via an approach called Model-Based (System) Architecting and Software Engineering (MBASE). His contributions to the field include the Constructive Cost Model (COCOMO), the Spiral Model of the software process, and the Theory W (win-win) approach to software management and requirements determination. He has served on the board of several scientific journals and as a member of the Governing Board of the IEEE Computer Society. He currently serves as Chair of the Board of Visitors for the CMU Software Engineering Institute. He is an AIAA Fellow, an ACM Fellow, an IEEE Fellow, and a member of the U.S. National Academy of Engineering.

Forthcoming events

Systems Available to All

Date: 19th September
Location: The Key Centre, University of Hertfordshire, Hatfield Campus.
Contact: David Shearer, Department of Computer Science, University of Hertfordshire. (d.w.shearer@herts.ac.uk)

The Disability Rights Commission has amongst its responsibilities to promote the equalisation of opportunities for disabled persons. If software developers ignore the needs and requirements of disabled users, could their employers be liable to prosecution? This meeting is a joint meeting between the BCS Disability and Requirements Engineering Specialist Groups to consider how developers can take into account the needs and requirements of all potential users.

RE-Calls

Recent Calls for Papers and Participation

First International Workshop From SofTware Requirements to Architectures (STRAW'01), May 14, 2001, Toronto, Canada.

http://www.cin.ufpe.br/~straw01

Held In Conjuction with ICSE2001.

Goals:
The goal of the workshop is to bring together professionals from academia and industry to exchange ideas, experiences to improve our understanding of the relationship between requirements engineering and software architecture. Topics of interest include:

- Requirements and Architecture modelling
- Deriving architectural description in concert with requirements specifications.
- Tracing architectural decisions to requirements
- Systematic derivation of parameter settings from requirements
- Dealing with requirements and architectural evolution
- Formal foundations and analyses
- Object-Oriented Requirements to Object Oriented Architectures
- Agent-Oriented Requirements to Architectures
- Education and Training: skills and traits for good requirements engineers and software architects
- Case studies and empirical studies,
- Tools/Environments for Requirements Engineers and Software Architects

REFSQ 2001 – Seventh International Workshop on Requirements Engineering Foundations for Software Quality, June 4th – 5th, Interlaken, Switzerland.

http://www.ifi.uib.no/conf/refsq2001/

Held In Conjuction with CAiSE*2001.

Goals:
The goal of REFSQ is to improve the understanding of the relations between requirements engineering and software quality. More precisely, REFSQ aims at having intensive discussion provoked by brief presentations about:

- solutions to known RE problems and shortcomings;
- innovative research ideas possibly initiating new research directions;
- industrial problem statements; and
- generalisations from individual industrial experiences.

Fifth IEEE International Symposium on Requirements Engineering (RE'01), Toronto, Canada, August 27-31m 2001

http://www.re01.org/

Scope:
RE’01 will provide an opportunity for researchers, practitioners, and students to exchange problems, solutions, and experiences in RE. It will emphasise the crucial role that RE plays in the development and delivery of systems, products, and services that permeate all aspects of life and increasingly serve users across national, cultural and professional boundaries. In addition to wanting systems to deliver required functions, users increasingly demand systems that are usable, reliable, secure and responsive. In a rapidly changing world, users and product managers expect today’s products to be adaptable to their future technical and social environments.

Submissions:
Details of the submission procedure are available from the conference website:

RE-Readings

Reviews of recent Requirements Engineering events. Reports by Ian F. Alexander

Panel Session on Requirements, City University, 17 November 2000

Neil Maiden welcomed a panel of 4 speakers to a requirements engineering panel session at City University. The format was a swift 15 minutes per speaker, and then an hour of panel discussion with questions from the floor.

Suzanne Robertson said that requirements were always about people. The waterfall model was false: phases overlapped, stakeholders interacted and had ambitions and goals. Requirement was a highly elastic word; as with all terms that become popular (such as Object-Oriented and Use Case), its meanings have multiplied, to include business objective, product goal, functional requirement, constraint, quality attribute, solution specification, and so on. To be useful for engineering projects, we need linguistic integrity. The Volere template enables specific use of terms, providing an interrelated project model containing documents, links and meanings.

Ian Alexander spoke about the use of scenarios to provide context for requirements engineering. There was no agreement on what a scenario was, but scenarios could be abstract or concrete, simple sequences or multi-threaded and branched. They could be represented as use cases, text, flowcharts, swimlanes diagrams, task models, goal hierarchies, or storyboards. They could equally well be worked out and dramatised as stories, sketches, or acted scenes in workshops. They could be used to elicit requirements, specify system behaviour, guide design, and form the basis of acceptance test cases.

Allan Kennedy presented Four Opinions About Requirements. A requirement was 'any statement with which compliance had to be demonstrated'. It had to provide an audit trail or connecting thread. It was a pervasive element, being used throughout the development life-cycle. And there was no single normative RE process. To support any number of processes tailored to specific projects, one needed a meta-tool able to handle a variety of models. For instance, requirements could be enabled to have dependencies, so that people could construct sub-requirements or could substitute one requirement for another.

Ian Graham argued for Extreme Modelling, on the pattern of Extreme Programming. Stories were good, so we could start with those. Testing was good, so we could aim for laziness, doing the least possible to pass each test. Interaction and dialog were good, so we could build teams involving developers and users. Iterative delivery was good, too. He said that Suzanne Robertson was wrong, the waterfall did happen sometimes, as when projects followed the pattern of getting a contract, doing work, and taking payment – the payment process was simply linear, even if the software development process was extremely non-linear. The Niagara Falls were chaotic, seen from anywhere closer than outer space. Key elements in RE included separating requirements from specification; traceability; scenarios; precision; and the development of a shared language between the people concerned. The Catalysis (formal) method aimed to meet all of these goals. A key principle was to delay assigning operations to types as long as possible, as these decisions were almost irreversible.

The **panel discussion** ranged widely. Among the central ideas were the world/problem versus machine/solution domain split; the need to manage different viewpoints, possibly politically as well as in engineering terms; the impossibility of defining a single unified process applicable to everything; and the individual and personal nature of elicitation techniques. RE involved human interaction and commitment; relevant skills were storytelling, acting, graphic design, interpretation, politics, mediation.

In Ian Graham's words, "Great software developers learn Shakespeare, musical instruments, 18th Century novels – not Computer Science. And they learn them by doing, not by sitting alone in a room."

The organizer and the speakers adjourned to a nearby restaurant to continue the discussions.

Scenarios through the System Life-Cycle, IEE/RESG all-day seminar, Savoy Place, London, 7th December 2000.

Ian Alexander (Independent Consultant) welcomed the seminar delegates to the IEE, and introduced the subject of system engineering with scenarios by giving a brief tour of the field. Scenarios could be understood in several ways, in particular along a scale from abstract (with classes of actors) to concrete (with named actors, times, and places); they could describe a more or less static situation, or could more typically include time by describing a more or less linear sequence of steps or actions involving one or several actors, which might be human or machine agents. Similarly, scenarios could be represented in many forms, from simple storyboards and cartoons through informal paragraphs of text such as Extreme Programming's 'User Stories', right up to carefully-structured sequenced or networked objects represented in text or as diagrams and permitting various kinds of automatic inference.

It was impossible not to mention the Unified Modeling Language, UML, which although reasonably laying claim to being a software (if not system) industry standard, had marked weaknesses as a medium for communication between 'users' and systems engineers. The use of the term 'Actor' to mean 'Role' was unfortunate. The use of a manikin icon to denote a role while supposedly not implying humanity was worse. And the claim that the visual sequence of Use Cases on a UML use case summary and relationships diagram did not imply any particular order of execution to possibly untrained readers was simply absurd. The whole purpose of using scenarios was to help engineers get a better idea of what 'users' actually wanted; any representation of scenarios meant to be shared with users therefore had to be simple, accurate, and not misleading.

Scenarios were not useful only during requirements capture, however. They could be used during specification to describe the desired behaviour and interactions of systems and subsystems. They could help engineers during design – often remote from contact with clients – to visualise the intended function of their subsystems in the wider context. And they could help throughout the system life-cycle in validating, predicting and finally in verifying the correct behaviour of systems: there was a close relationship between a scenario that described what a system ought to do and what results it ought to deliver, and a test case that checked that the system in fact did what it should. Furthermore, scenarios could be animated, simulated or otherwise executed in various ways to demonstrate and test correctness even before systems had been designed.

Neil Maiden (City University) said that he was interested in measurable tests of systems that could be related to scenarios, in other words in Fit Criteria. He considered that scenarios were 'middle-level abstractions for talking to stakeholders'. He wanted to find a way of actually 'specifying use cases, not in a horribly waffly way a la Jacobson'. He had worked together with BAE Systems to construct a simple practical mechanism. Part of this involved 'begging, borrowing, or stealing Exception Classes (i.e. a Taxonomy of Exceptions) from academia, HCI, Systems Engineering, Cognitive Psychology or wherever'. The result was over a hundred classes of exceptions, some of them quite general, some of them specific to the military domain, many of them new to systems engineering.

David Corrall (BAE Systems) then took over the podium and explained how he had tried to marry requirements with scenarios. The idea was to make a mutually consistent structure relating objectives, functional requirements, constraints, and contexts. This was a far bigger undertaking than the simple list-of-events level of scenario description that most people usually considered, he said. He presented a diagram showing a ladder of information structures, from the Concept of Operations at the top on the System/Requirements side, down through Domain Tasks, Concept, Functions/Constraints, Organisation, and Solution. These roughly map to business processes, business requirements, user requirements, system requirements, design, and coding, though evidently few non-military systems go into anything like so much detail. On the operational context side, these structures are mirrored by Operations at the top, going down through Situations (activities), Incidents (at system level), Events (at subsystem level) and finally Event Parameters, related to the solution. This project architecture enables the performance of whole platforms to be studied and optimised, based ultimately on the behaviours of all the equipments that will be fitted. For instance if you were fitting a sensor, a network, and a weapon together, then a scenario that involved detecting something with the sensor and responding by passing information over the network to the weapon would have a performance limited by the delays of all three components. Neil Maiden added that to scale up to handle large systems you had to be systematic.

Janos Korn (visiting fellow, LSE) said that to judge from the preceding speakers, there was tremendous diversity of approach to scenarios. He introducted two concepts: the idea of a property (or attribute), and the rule of conservation of state – nothing can change its own state by itself. These simple ideas enabled one to turn complex linguistic statements into sets of simple statements, from which one could in theory derive specific solutions (such as software designs). One could also attach degrees of intention rather like probabilities and then (again in theory) calculate results using Durkin's Certainty Theory. One could then predict outcomes, identify gaps in scenarios, and hence generate candidate requirements, as in Neil Maiden's work. However this had not yet been automated.

Deborah Hopkins-Hurt (Police IT Organisation) said that she had been involved in trying to describe a blueprint for all police business. The Rational Unified Process and the Unified Modeling Language (UML) formed a good basic place to start if you didn't know what you were doing. The PITO was responsible for writing requirements for third-party developers. Its work therefore consisted of business modelling, problem identification, requirements gathering, and finally requirements specification. 'We wrote simple sentences in a Cockburn-like way and then had walkthroughs of the scenarios with the business experts' she explained. 'We prepared simple how-to guides using Cockburn's book *Writing Effective Use Cases* as advice. 'We didn't use a lot of UML notation except activity (workflow) diagrams', she said. Rather, the team put together simple use case descriptions.

John Hughes (Lancaster University) spoke about his use of ethnography (or in today's parlance, fieldwork) as a way of allowing the social dimension to inform design. He had (famously) worked on air traffic control as well as for the police, aerospace, a retail bank, and for such diverse applications as risk in the health service and the nuclear industry and the domestic context in which consumer electronics are actually used. Fieldwork was the only way in which one could go beyond what people say they do – more or less abstracted, verbally described claims – to what an observer could see and hear. It took real life as a real-time actual social phenomenon and described how work got done. It was 'not a theory-rich activity'; good fieldworkers knew to 'leave the theories at home'. Concepts that seemed clear enough to everyone else like

Task and Goal became problematised, became activities to be investigated. Fieldwork had limitations; problems included the scale of work that could be attempted, time pressure, and communication between different disciplines with their own ways of speaking and thinking. A fieldworker 'always knows more than can be written or spoken'; this knowledge 'is lost if the worker is run over by a bus'.

There were definite parallels between ethnography and scenarios, both being about the social dynamics of a domain. Scenario elements in ethnography included the setting, agents/actors; objectives or goals; a story with a plot, being a sequence of actions or events; and the exploration of current or future possibilities. Debriefing was crucial. Stories were best told traditionally with a beginning, a middle, and an end; the 'day in the life of' format was effective, and enough detail (not Proustian, though) to bring out the nature of the real work in its setting. One could make an "ego-logical" division, i.e. from the point of view of individuals participating, or an ecological one looking at niches and settings. The result was multiple views of the work in question.

Neil Maiden commented that no-one had really looked at what was meant by abstraction in scenarios. Very probably we used several types of abstraction in combination. He wondered what sorts of questions ethnography could answer, and what it could do that nothing else could.

John Hughes said that understanding of settings, resonating with human use, and social context were ethnography's main contributions. It was possible that what business analysts did came close to fieldwork, but they came armed with frameworks, whereas the fieldworker came with nothing beyond a wish to look at (social) dynamics. Interviewing and observation of small vignettes of real life, micro-scenarios, combined to build up day-in-the-life stories. These could be used to envisage where systems could be needed. Fieldwork was thus a resource for design; it counteracted the 'over-engineered technological methods that themselves didn't work', he argued, concluding 'You often get surprises'.

John Potter (BAE Systems) spoke about the problems facing all future large-scale projects: complexity (the reason for Systems Engineering), change, and ever-faster performance. System requirements could be quantified by scenarios, as these related different parts of the system's behaviour, permitting estimates to be made of future end-to-end performance. Scenarios had to be designed to exercise as many system functions as possible, in the manner of test cases. At operational concept level, animating scenarios could be useful on many projects; BAES had a simple portable PC tool that supplemented more detailed performance modelling, enabling them to gain quick agreement between users and analysts. The scenario in his approach related together four other elements of the specification: contents; purpose; life-cycle; and form/structure. It led to faster agreement, better and fewer meetings, sharper requirements, and earlier feedback from prototypes.

Karl Cox (Bournemouth University) quoted Colin Potts of Georgia Inst.Tech.: 'A scenario is a proposed specific use of a system'. He had made simple use of scenarios with a City institution, halving the time needed to agree requirements. His use case were straightforward lists of actors (classes of user) and their actions. These could be drawn as simple flow diagrams, with no alternatives or exceptions modelled. Even such a basic approach proved very helpful, mapping prototype user interface events on to scenario events.

In the **Panel Discussion** that followed, **Neil Maiden** said that in a CREWS project survey, 80% of firms used prototypes alongside scenarios. He wanted to know how other requirements acquisition tasks should be combined with scenario use, and what questions ought to be asked. **David Corrall** argued that 'the system is the requirements', and these had to be structured by the scenarios and used alongside them. **Karl Cox** said that the power of the business analyst was reduced by the existence of scenarios; they were a democratizing force. Neil Maiden pointed out that safety and hazard analyses were often used with scenarios; with usability analyses, one was into HCI, while with reliability one was into metrics and modelling, – separate schools of RE. The panel discussed the value of scenarios in enhancing co-operative work, taking RE away from 'throwing a specification over the fence' towards iterative and parallel development of user and system requirements and system architecture in the style of concurrent engineering. More education in the style of today's seminar was, in **Andrew Daw**'s view, urgently needed; user and system requirements needed to be seen as peer documents, not as stages in functional decomposition. Even the architectural design was not necessarily more detailed – if some equipments were mandated (as COTS components) then trade-offs were limited. In other cases one tried to maximise design freedom and trade-off space, avoiding over-specification.

It seemed that everyone enjoyed the day and felt they had benefited; there was lively discussion over the excellent lunch and during the breaks. And a delegate commented that scenarios and systems engineering ought to feature more strongly in future IEE/RESG programmes.

RE-Papers

Requirements Engineering - the new frontier in competitive advantage?

Barbara Farbey
University College London

"Competitive Advantage" has become the mantra of our times. As the old advertisements used to say, there are many ways to use this product: MBA students use it to demonstrate that they have read their lecture notes;

managers use it to justify any pet project from re-organising distribution to betting the company on a new strategic alliance; consultants use it for whatever it is they want to sell today. Academics, who ought in all conscience to bring a little *gravitas* to all-consuming trends, have spun off "co-operative advantage" and "co-opetition" as separate industries of their own.

Is there anything to competitive advantage? And are there any areas that are likely contenders for the next wave? In brief "Yes. But ..." and "Requirements Engineering" .

"Yes" because if a firm can outperform its competitors in ways that are important to the market then, other things being equal, it stands a good chance of picking up business.

"But ... " because unless the advantage is really inimitable, the competition will eventually catch up. Consumers will be the only long-term winners and the grounds of competition will shift.

For the rest, if the skill, process, technology, competence or whatever it is that is providing the advantage is imitable, there are fundamentally three positions you can be in. These three positions are defined with respect to the current technological frontier, that is, the best anyone can achieve with the current technology and ways of doing things.

First, you can be lagging behind your competitors. Your first priority here is to catch up: there is no longer any competitive advantage that accrues to a retail bank by installing ATMs. You have to have ATMs, or an agreement to use someone else's, to be a competitive bank on the High Street.

Second, you can be technologically in contention, but want to shift the balance of resources. E-commerce is likely to provide an example. As the larger, well-established firms move into e-commerce, they have the resources to strike a strategic balance between all the diverse channels open to them. They can move into e-commerce, while still keeping a greater or lesser presence in traditional channels. The exact balance will depend on the market, the firm's efficiency at tracking it and how they see their own internal capabilities.

Third, you can attempt to change the rules of the game using new technology. In this day and age that usually means information and communications technology, but it could be a different way of organising, as in the new supply chain management. If you can break through the existing technology frontier, you undoubtedly have a potential competitive advantage. That is why there is so much emphasis on innovation. Constant innovation may be mind-blowing, but it holds out the promise of survival - at least until everyone else catches up.

One area of expertise that currently fits between "essential to compete" and "breakthrough" is Requirements Engineering. Requirements Engineering is a branch of software engineering. It is concerned with understanding in an orderly way what is required from a computer system by its stakeholders. Requirements Engineering addresses the processes of requirements elicitation, definition, modelling, analysis, specification and validation. It includes questions of how these processes and the resulting requirements are to be managed.

Requirements Engineering offers a fresh approach to systems analysis, building on the success of the engineering paradigm in industrial design. It argues for a systematic approach to gathering and managing requirements, separating the "what" from the "how" of design, introducing a rational search for alternatives and testing prospective solutions.

Its potential as a strategic technology exists because designing and developing information systems that meet the ongoing requirements of their users is not universally well done and the relevant skill sets are hard to put together. Research estimates that more systems than not fail to meet the requirements of their intended users. Some estimates put the figure as high as 85%[i]. In this view recent high-profile systems failures ranging from the Passport Office to the Royal Opera House are not isolated occurrences. They are just the latest in a predictable and dismal catalogue of similar failures

Given that IT and systems absorb huge expenditures: 1-2% of general corporate expenditure, 16% in the financial industry[ii] and more than 50% of government departmental and agency spending[iii], this failure to meet requirements is a cause for genuine concern - and an opportunity to do better. The frontier is wide open.

Paradoxically the root causes of failure are well known, well researched and well documented. They are not necessarily technical, although there are some very significant and challenging technical problems. Most of the problems can be traced to a failure either to consult users widely on what they really need from the system - their requirements - or to establish basic requirements in the early stages, when they can more easily be incorporated into the new system. The later the change, the more difficult and expensive it becomes to change the system.

To deal with the problems we need to find answers to three questions: What do we need to do to establish user requirements and needs? How do we do it in a timely and effective way? And how do we persuade systems designers and users to do it at all? Or in other words how do we institutionalise design and development practices, with respect to requirements, that will lead to more usable and useful systems?

The European response has been co-ordinated by a Network of Excellence in Requirements Engineering - RENOIR[iv]. 85 leading research groups from industry and academia make up a collaborative research

---

[i] Clegg et al
[ii] PriceWaterhouseCoopers, Annual Survey, 1998 figures. http://www.pricewaterhousecoopers.com
[iii] PriceWaterhouseCoopers, Annual Survey, 1998 figures. http://www.pricewaterhousecoopers.com
[iv] RENOIR http://www.cs.ucl.ac.uk/research/renoir

network, addressing a range of issues, including crucially the organisational context, the groundwork necessary for Requirements Engineering and the communication of results, as well as acquisition and analysis of requirements.

In the UK the British Computer Society's Requirements Engineering Specialist Group (RESG[v]) is active in promoting Requirements Engineering, with a strong focus on the industrial application of leading edge research. RESG holds regular meetings and maintains an active website. The site contains details of RESG events and related conferences, publications and other resources. There is also a regular newsletter for RESG members, with back numbers accessible from the site.

All this adds up to a considerable resource. To convert the resource into workable, acceptable processes and procedures - to institutionalise it - management needs to take on board three issues: value, diversity and change.

First, the value of Requirements Engineering needs to be established. In its initial stages, Requirements Engineering looks very like bureaucracy. It costs money. In the rough and tumble of everyday organisational life anything that looks bureaucratic will get short shrift unless the arguments for it are overwhelming. But we have learned painfully over recent years of the value of systematising and bureaucratising quality procedures. Requirements Engineering will improve quality, but we have little idea of the figures. Academics would do well to turn their attention to the costs, risks and benefits of the requirements process.

Second, we need to learn how to work with very soft requirements and many diverse constituencies, or stakeholders as they are commonly called. There is a story about three baseball umpires and how they call balls and strikes. "The first calls them as they are, the second calls them as he sees them; the third replies that 'they ain't nothing 'til I call them' "[vi]. Often, very often, requirements are soft, they "ain't nothing 'til someone calls them". This is hard for users to understand, and even harder for the software engineers. Their training and self- image militates against it. But only those engineers who thoroughly understand the many constituencies for a system and how they interact to produce a requirement are able to produce a system that the users will use.

Third, we have to recognise that in most organisations the Requirements Engineering approach will necessitate organisational change, and be prepared to manage that change.

For the moment, organisations that do well on all three counts will be ahead of the field. Moreover, those that manage their constituencies well will have a capability that is personal and hard to imitate. They will have a competitive advantage - until good practice in Requirements Engineering becomes universal. That would indeed be a breakthrough.

References and acknowledgements

- Clegg, C., Axtell, C., Damodaran, L., Farbey, B., Hull, R., Lloyd-Jones, R., Nicholls, J., Sell, R., and Tomlinson, C. (1997). Information Technology: a study of performance and the role of human and organizational factors, *Ergonomics*, 40, 851-871
- Arrington C.E. and Francis J.R. Letting the chat out of the bag: Deconstruction, privilege and accounting research, in Calas M.B. and Smircich L. *Postmodern Management Theory*, Ashgate, Dartmouth, 1997, Chapter 12

Conversations with:

- Charles Baden Fuller, City University Business School
- Anthony Finkelstein, University College London
- Frank Land, London School of Economics
- David Targett, Imperial College London

What is RE Anyway?

Ian Alexander
Independent Consultant

A recent discussion on the SRE mailing list focussed on what RE is actually about, and whether the Engineering referred to has any meaning, or is simply a noise-word like 'handling' or 'management'. Practitioners and academics have been arguing to and fro on the question. In other words, we are not sure that we believe that RE is engineering.

To put the issue in context, let me tell a true story. A friend of mine has a houseboat on the Thames. A compartment is reached by a hatchway, protected by a heavy hatch - 15 kg of it, to be precise. From above, this is not too bad, but from below, the task of climbing out while carrying something like a tin of paint in one hand and a brush in the other was pretty difficult: my friend used to lift the hatch with his head and then wriggle out.

One day, he came across the solution - an advertisement for air-springs. The air-spring is an ingenious contraption of the sort used to make car boots open easily. He phoned the makers and they asked for exact measurements of his hatch. By return of post came a precise drawing - a computer print-out - showing where to fit the air-spring attachments, and which size of spring to use. He followed the totally counter-intuitive recipe exactly, and now lifts the hatch with one finger.

Well, that's engineering all right. An input specification ('A hatch of 15 kg, dimension x´y´z …') is transformed completely algorithmically to an output specification ('model AS-123, attached at x1,y1 and inserted at x2, y2'). The result is ingenious - engine-ous, to spell it out - in the best sense; the problem is totally and permanently solved by calculation of the behaviour of an 'engine' of a special kind.

[v] RESG http://www.resg.org.uk
[vi] Arrington and Francis 1997

Trouble is, our discipline doesn't have nice sharp input specifications. In fact, it often doesn't have any input specifications at all. Where classical will-the-bridge-fall-down engineering is all about transforming specinput into specoutput, RE is about transforming specfuzzy or absent into specinput. Our product is the very same input specification that your classical engineer would (we hope) recognise as the starting point for real engineering.

If that really is where engineering proper starts, then obviously whatever we do may well be useful and necessary but by definition isn't engineering. Why then do we claim the title of engineer? One answer is that it's just to acquire the gloss or sheen of reputability, a little of the reflected glory of Henry Rolls and Isambard Kingdom Brunel, while inwardly we feel we are little better than quack psychotherapists, taking our clients' money for some half-understood advice and doubtfully efficacious patent nostrums.

But we need not be quite so hard on ourselves. That input specification has to be a solid, reliable, accurate, objective analysis; in short, a well-engineered product. The process by which it is obtained and formalized cannot be like the calculation of a beam's thickness or an air-spring's insertion point, because it is a different kind of engineering. Ours is a discipline that efficiently and rigorously elicits, organizes, checks, measures, prioritizes and documents what a set of diverse stakeholders want - and helps them to agree on the specification of a solution. Or at least, that is certainly what we ought to be doing. Sometimes, the specification may be quite formal; but I would argue that by the time the problem is well-enough understood and scoped to be formalized completely, the RE task is as good as over; the hard part is getting something solid from something nebulous. A thermal engineer might think of it as condensing a gas - the speeding random requirements all end up crystallized, lined up in neat rows. As for requirements management, that is the easy bit - putting the elicited items into a neat database to enable projects to keep track of them safely - but it is certainly part of the overall RE process.

Consulting engineers have always helped their clients to state their needs accurately, as the basis for the calculations that followed. But perhaps they have not paid full attention to how this subtle, often informal human process was conducted. Highly expert engineers have probably always elicited requirements almost unconsciously. But that effortless-looking ease should not delude us into thinking that knowledge, practice, and experience were not needed to acquire the requisite skill. In fact, recognizing only the most obvious calculation-based skills as engineering is rather like searching for your car-keys under a lone streetlight because that is the only part of the street bright enough to search without effort.

CORE-Blimey

The last in regular column by Geoff Mullery, of Systemic Methods Ltd.

Swal S’notwen

In past contributions I have mentioned that the major reasons computer systems projects fail are not technical. One regular reason for failure is the incompetence of the project’s parent organisations (in both customer and developer sides) in management of technical development. There are other reasons for failure, but it is this one I shall address here.

Let me start by postulating a typical project situation for the type of project which regularly fails. To avoid the possibility of lawsuits names used in the example have been artificially selected, but hopefully you can see the parallels.

You are a project manager in a company newly set up by a large parent organisation to bid for a contract developing software for a space transport mission. The software will control transfer of supplies between Earth and colonies on the Moon and Mars. You have little background in the mathematics necessary, but you have heard that there are three competing approaches.

The traditional approach, defined years ago by someone called Ptolemy has a reputation for being messy. The calculations voluminous but the maths is fairly trivial, revolving around look-up tables and computing paths round circles. You can get lots of staff who know about look-up tables and circles. They claim their way works because it was used on transport projects for years and some of them worked, albeit with a tendency to arrive with only one wheel left on the wagon.

There is a more recent mathematically oriented approach by someone called Newton. This is a more elegant approach, with fewer equations, but it has known inaccuracies and requires people trained in a peculiar notation called calculus. It was quite popular among academics for a few years and now has a strong following in industry. It has been used successfully on a project which calculated how to get a bicycle from Guildford in Surrey to Reading in Berkshire – provided an upgrade was made, to provide a small engine.

The third and most recent approach was defined by someone called Einstein. It is very popular in academia but requires a great deal of training, not only in calculus but also incurs the need to understand strange new terms like time dilation, quantum physics and the uncertainty principle. It can only be applied by experts and they can show you the approach work using a model based on marbles and short bits of string whose dimensions they calculate with great accuracy.

You must select one of these approaches for your bid and your marketing department tell you that it must not use more than twenty man years or an elapsed time to complete greater than two years. Your management is reluctant to spend money on large-scale training courses, but will send two staff on a course provided it lasts less than a week. They will then have to train the rest of the team as the project proceeds.

You can hire five days effort from an expert external consultant to help prepare the bid – from the agency your company normally uses. Their advertising brochure says they have very well qualified mathematicians on their books. Of the three candidates sent it turns out that each strongly advocates a different one of the candidate approaches (the agency knew the approaches you were interested from your description of what you were after).

Each candidate tells you about project horror stories with the other approaches – a cost over-run and inaccuracy for a Ptolemy project or a Newton training course that left attendees only able to develop systems for movement via bicycle between towns less than fifty miles apart or massively expensive Einstein training followed by a three year wait while the team worked out the most precise solution for the motion of the marbles accounting for the elasticity of short strings.

You judge that if you don't apply one of the two more modern approaches you won't get the contract. The project has been judged not safety critical (if the transport is destroyed it only sacrifices the crew, as long as the accident doesn't happen near a colonies or the Earth) so you avoid the worse training costs of the Einstein approach and adopt Newton.

Your two staff members attend a Newton course, given by someone well qualified, because it is not long since s/he went on the same course. They come back with slightly less knowledge than their tutor, but they are convinced that the approach is logical and there should be no serious problem. The bid is entered, with a claim that project staff are trained in the Newton approach and a promise to finish on the time and cost profile marketing believes the customer wants.

You win the contract, so work gets under way. Your Newton-trained staff are neither managers nor trainers and are primarily interested in doing the technical work. They do as little training as they can, trotting out their imperfect understanding when they do and concentrate that understanding on actual technical development.

The other staff don't understand what they have been "taught", see little sense in it and try to apply the Ptolemy approach they are used to, but make it look like the Newton approach when they write the results down. The trained staff do the quality control and can't understand why their own stuff doesn't seem to be working, but assume that is just a temporary problem so they concentrate on trying to overcome the quality nightmare the other staff are turning out.

They eventually realise the only way to make progress is to do all the complicated stuff themselves, using the other staff only for trivia. Unfortunately there are many more complex problems than trivial ones, so they, who comprise 10% of the effort end up trying to do 90% of the work.

To make progress at the planned rate you agree with the customer to make deliveries, starting with the easiest bits first and leaving the complex bits to the end. Your customer is suspicious but is not really competent in the maths either, so assumes that you know best – and anyway, your company carries all the risk.

About two thirds of the way through the project the disaster can no longer be hidden from your company's management. Your Newton trained staff resign to join a company which is seeking staff experienced in the Newton approach – and they now have a year's experience!

An emergency meeting reverts the whole project to the Ptolemy approach, because no one left understands the Newton approach and there is no chance of progress any other way. All the difficult stuff which was near to working was produced by the departed Netwon "experts" and now needs translation into the Ptolemy approach so that the staff now available can understand it.

This means that the customer's QA will detect that the project is not using Newton for any part of the project, so the customer must be told. Fortunately the contract only declared that you would *use* the Newton approach – not that it was the only approach you would use, nor that what you would deliver would use the Newton notation (yes, I really have seen this stunt pulled!).

The contract only said you would adhere to the company's quality system and there are Ptolemy notation standards and procedures in the company quality system as well as those introduced specially for the Newton approach. The customer is very upset, but grudgingly accepts the situation because you assert that the re-design will make things get back on course and the quality will improve. Unfortunately translating from Newton to Ptolemy is very difficult since the staff don't understand the Newton approach. In fact they make an even bigger mess and the project gets to the stage where it clearly can't reach Mars and probably won't reach the Moon in the time allowed for completion.

Now you are forced to look for customer errors to provide your company with a contractual excuse for the failure. The best excuses account for only a small proportion of the failure, so it is clear that the customer is likely to get a large amount of money back by invoking penalty clauses.

This is unacceptable to your company's management, so they dismiss you and inform the customer that if penalty clauses are invoked they will wind up the company, with the customer as one of its major creditors (yes, I have seen this one tried too!). This means the customer will be left with no system and Moon and Mars colonies which can not be supplied in the foreseeable future.

The customer agrees to a reduced project which will reach only the Moon and six months later than planned. The customer plans to issue a new invitation to tender for a Mars transport developed based on the Moon transport project, to complete a year later than was planned for the original project. The reduced project runs to completion – but is eventually nine months late and can deliver only 75% of the stores required. Luckily the Moon colony can survive that long.

The follow-up Mars project goes out to tender and after the bid assessments it turns out that your old company has the most compliant bid for the least cost and is judged to be commercially credible since it has had more experience with space transport projects than any of its rival bidders. Hence they win the contract and the merry-go-round starts another rotation (is this the spiral model in practice?).

Until one or more objective frameworks is established for assessment of development approaches, not only in theory but also in practical use across a range of application domains and with demonstrable statistical validity of its measurement and evaluation methods there will be no answer to the problem illustrated in this story of organisational incompetence in the critical area of technical management.

If a new project were to start tomorrow, with the aim of producing a space transport system based on ***Newton's Laws***, as in the example I have given here it should be of no surprise to any of us if, as suggested in the title of this contribution, its project management approach gets the whole thing back to front.

***RE*-Publications**

Book Reviews

All books reviewed by Ian Alexander

Software Engineering (6th Edition)
Ian Sommerville
693 pages, Addison-Wesley, 2000
ISBN 0-201-39815-X (Boards)

It is always a pleasure to come across clear, direct, and practical engineering writing. This book is all that, and humane too, with simple coverage of professional ethics, and exercise questions on the value of human life in safety-critical systems. Sommerville is already justifiably well-known for the earlier editions of this book, and he has excelled himself in this new edition. He has attempted the near-impossible: actually to shorten the book. This effort has been successful not just in reducing the page count but crucially in strengthening the thread of argument, making an already good text even stronger. This has been achieved in the face of tremendous growth in software engineering as a profession, and a burgeoning of knowledge – for instance in distributed systems. It is hard now to recall that just five years ago most businesses used the Internet only for email – or not at all.

The book has been restructured in line with its new balance of topics. Part 1 looks at the whole process in four rewritten chapters. These include a genuine FAQ responding to the barrage of email the author received, but not (as he says) the pleas of students for help with their homework. Students do get, however, an admirably broad discussion of the problems that software engineering faces at each stage: for example, workers may resist implementation of systems that reduce their responsibilities or threaten to reduce the number of jobs. Old-time computer science textbooks never mentioned anything as messy as that.

Part 2, called simply Requirements, shows the student how to put together conventional user and system requirements documents, and how to make good use of models, prototypes, and if need be formal specifications.

Part 3, the core of the book, looks at Design in six chapters. The focus on distributed, object-oriented, and real-time is justified: many systems are all three, and most systems of any size are at least one of these. The emphasis on reuse is also welcome.

The remaining parts, each of about three or four chapters, show the considerable shift in emphasis from the previous edition: Critical Systems, V&V (now merged), Management, and Evolution.

It is striking how much of this is about processes and how little about the details of specific practices. As Sommerville states in the Preface, "I assume that readers have a basic familiarity with programming and modern computer systems and knowledge of basic data structures such as stacks, lists and queues".

Requirements engineers will find improved coverage of important areas. Sommerville gives a straightforward account of functional and non-functional requirements, system and software requirements documents, and in a separate chapter of the processes of RE. In addition, in the later chapters, Sommerville covers behavioural modelling, prototyping, formal specification (in less detail), major constraints including safety, dependability, and availability, and change management. In short, the book covers both RE and Systems Engineering to a reasonable level, as well as introducing the basic concepts of software design and testing, project organisation and management.

Software Engineering thus has a much wider scope, and is far less concerned with computer science, notations, or algorithms than earlier accounts of software engineering such as, say, Sue Conger's admirable and practical 1994 textbook (*The New Software Engineering*, Sue Conger, Wadsworth 1994, ISBN 0-534-17143-5). Sommerville has in other words moved towards a systems engineering view, and this is explicitly covered both in a terse 20-page introductory chapter on 'Computer-based system engineering' and in clearly argued sections on issues as diverse as modelling, architectures, critical systems, and process improvement.

Teachers should be pleased to discover that the website at http://www.software-engin.com provides an instructor's guide, slides for each chapter, Java source for the examples, as well as examples from previous editions on formal specification and programming in Ada and C++, these topics having been greatly reduced in the 6th edition.

Students of software engineering and computer science will want their own copy of this major and much enhanced textbook. Teachers will find it an invaluable resource. Researchers and practitioners in systems and software engineering will find it a useful aide-memoire.

Writing Effective Use Cases
Alistair Cockburn
270 pages, Addison-Wesley, 2001
ISBN 0-201-70225-8 (Paper)

Use Cases are the object-oriented system designer's way of thinking about requirements. Rather than treating functions as separate items, a use case groups together a related set of requirements as a readable structure of scenarios under the heading of a single clear goal.

Cockburn is probably the world expert on actually organizing requirements into use cases. He has a clear and incisive writing style and a determination to get to the core of any problem.

This is a really clear, practical and above all simple account of how use cases can be put to work to specify system functions. It is not a complete textbook of requirements engineering:

"use cases .. are not all of the requirements – they are *only* the behavioral requirements but they are *all* of the behavioral requirements. Business rules, glossary, performance targets, process requirements and many other things do not fall into [this] category."

The book is based on a wealth of experience, and offers much help to the beginner as well as to the practitioner. The focus is sharply practical but there is a short and helpful list of referenced books, papers, and online resources. Cockburn is especially good at stating what is not yet known and what needs to be explored further.

The book is neither about UML nor a critique of it – as the author explains, most of what he has to say fits inside those little ellipses and is therefore compatible with the language. This is not to say the UML concept, or popular accounts of it, are in Cockburn's opinion perfectly formed. In fact:

"…if you only read the UML standard, which does not discuss the content or writing of a use case, you will not understand what a use case is or how to use it, and you will be led in the dangerous direction of thinking that use cases are a graphical, as opposed to a textual, construction."

The diagram notation of the UML tends to divert attention away from the text; the subtle distinction between generalisation and extension is not particularly suited to describing desired behaviour; and the language is poor at describing how use cases relate to each other (though other diagram types can indicate temporal relationships). What is more, though Cockburn does not say so, some of the terms chosen in the UML seem simply unfortunate; an agent, whether human or machine, is known as an Actor, and represented by a manikin figure – at best, rather a confusing choice of symbol. Cockburn points out some of the difficulties, accepts that they are here to stay, and works around them by concentrating mainly on use case text. Specifically:

"The UML use case diagram, consisting of ellipses, arrows, and stick figures, is *not* a notation for capturing use cases. The ellipses and arrows show the packaging and decomposition of use cases, not their content. Recall that a use case names a goal… the ellipse diagram is missing essential information, such as which actor is doing each step and notes about the step ordering…"

Amen to that.

The book is neatly organised into Introduction, three Parts, and some useful Appendices. There is an effective Index.

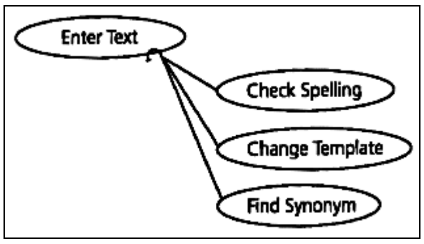
The Introduction explains what a use case is "(more or less)", what they are good for, and how to get started.

Part 1 looks at the Use Case Body Parts – the contract for behavior, scope, stakeholders and actors, three named goal levels, preconditions, triggers and guarantees, scenarios and steps, extensions, and use case formats.

Part 2 looks at Frequently Discussed Topics, including completeness, scaling up to large systems, CRUD and parameterised use cases, business process modelling, linking to detailed requirements (including constraints), use cases in the overall process, extreme programming, and common mistakes (especially, too much detail, too low level, and designing the user interface).

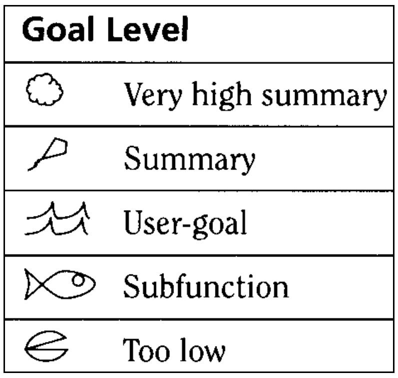
Part 3 provides Reminders for the Busy, including reminders for each use case, for sets of use cases, and for ways of working.

Appendix A takes a dozen pages to explain use cases in UML, complete with the difference between the *includes*, *generalises*, and *extends* relations. The reader is recommended to take advantage of the rules of UML to customise the arrow styles so that the different relations appear visually different. For example, *extends* can be drawn with a coathanger-like hook connector so it doesn't look like *generalises* – too many people blindly assume that they must use the default set of symbols, and produce unreadable documents as a result.



Cockburn ends with the advice "to spend time writing clear text, staying away from *extends*, and not worrying about diagrams." The other Appendices offer answers to some of the exercises, a glossary, and a short list of references.

Cockburn organises use case text in many ways, supplementing it with icons to indicate black or white box views and goal level (cloud, flying kite, user-level ocean surface, underwater with the fishes, or 'too low' down amongst the clams). He has devised a range of templates of differing degrees of formality, from casual to 'fully dressed'.



Goal Level	
☁	Very high summary
🪁	Summary
🌊	User-goal
🐟	Subfunction
🐚	Too low

For comparison, the Rational Unified Process (RUP) style is illustrated; "I find the use case quite self-explanatory" comments the author, though we may add that, like his fully-dressed style, it is plainly a lot of work to document a use case in this style – Ian Graham tells the story that a corporation discovered it took 2 weeks per case; since they had 1500 use cases to write up, they abandoned the attempt to follow the RUP.

Perhaps the aspect of the book that is most open to criticism is the suggested applicability of use cases to everything from low-level subfunction specification to business process modelling – it certainly sounds a strong claim. Cockburn devotes an entire chapter to the subject of business process modelling. He suggests that

"often there is no time [to] write white-box business use cases .. without mentioning the new technology"

followed by writing

"black-box system use cases that show the new system fitting into the technology-free white-box business use cases".

The solution is to dive right in to specification:

"a light business process documentation will be generated as a normal part of the system requirements exercise."

This approach may reinforce the worst fears of UML critics, who argue that the whole approach is back-to-front, starting as it does with an assumed system and working outwards towards 'users' whose function is to serve that system. Certainly one would not want to develop a business-critical system on that basis.

Cockburn's perspective is admittedly that of a skilled developer, but he is aware that other people who are not developers exist, and that system scope is a critical point of agreement.

I wholeheartedly recommend this book to everyone involved in defining what systems ought to do for their stakeholders. It is streets ahead of all other accounts of use cases. If the story is still not perfect, and if some of the claims seem a little too strong, the book is state-of-the-art and the experience genuine and practical.

Use cases will not go away. They represent a growing recognition that scenarios ought to be identified and recorded. They also indicate a definite movement in thinking away from coding towards specification, and, with their emphasis on users, towards stakeholders and their requirements. These trends are entirely to be welcomed.

Students – and their teachers – will find this a useful source of ideas, though they should take care to find out about alternative views on requirements and scenarios. Practitioners of object-oriented development will need no encouragement; engineers used to more traditional styles of development will find the book stimulating and thought-provoking.

Handbook of Action Research Participative Inquiry & Practice
Edited by Peter Reason & Hilary Bradbury
Sage, 2001, ISBN 0-7619-6645-5
468 pp (boards), Price £69.00

This is a tremendous piece of work: scholarly, concise, even-handed and wide-ranging.

What is Action Research? The idea is that research (or software engineering), instead of being remote from the subject(s) being investigated, should involve them directly. It should therefore be a participative undertaking, closely concerned with practical action rather than done for academic advancement. This leads to a democratic political standpoint, where people are equal in value, and where practical results are a better measure of effectiveness than the number of times papers are referenced.

Action Research, in other words, has a revolutionary agenda. This does not stop each chapter of the book from having a complete academic apparatus: a page at least of references, a densely-argued narrative, appeals to authority, statements of position.

What is wonderful in this book is that for the first time – *pace* Uwe Flick's excellent *Introduction to Qualitative Research* – a single volume provides a readable comparative tour of the main strands of Action Research, written by the practitioners themselves. The editors have achieved the prodigy of marshalling the many authors into a harmonious sequence. Each chapter is strictly limited in length to ten carefully checked and formatted pages.

The book is divided – after the introduction, which is itself a state-of-the-art review article of a dozen pages – into four parts, each of about a dozen chapters, entitled Groundings, Practices, Exemplars, and Skills. I will just touch on a few of the themes.

Participatory action research may sound exotic, but every project, consultancy and training course is in fact a collaboration whose success depends on every participant. Happily there is at least a modest movement towards participation in business and IT. For example, Use Case analysis admits the central role of usually-human 'actors' in every scenario and requirement. Similarly, Action Research emphasizes that people know things by doing.

To make systems that really work for people, then, as Robert Louis Flood writes, we must move from reductionist analysis to 'systems thinking', a concept that includes social awareness, sound methods, and political fairness. Flood himself, with Michael Jackson and others, has done much to bring this sort of systems thinking into the world of Information Systems, though it remains quite technocratic – and requirements engineers still have a lot of work to do to bridge the gap between 'users' and 'systems'.

Several authors reflect on the methodology itself. Edgar Schein helpfully constructs a taxonomy of types of research based on three axes: the degree of researcher 'involvement'; the degree of client involvement; and whether the researcher or the client initiated the project. For instance, where the researcher/consultant starts the project but no-one is heavily involved, you get dry demographic studies (statistical analyses of data). Where both are fully involved, you have Action Research. Conversely, where the client calls for the project, and both consultant and client are fully involved, you have process consulting. All these modes, and others that Schein identifies, make excellent sense in disciplines such as requirements engineering.

Helps include a splendid index, which distinguishes no less than 30 categories of 'social'; interesting notes on the contributors – another tour of the field, as many leading experts are represented; a preface which is the book in microcosm; and a condensed-reading chapter consisting of 10-line outlines of each contribution.

This is a textbook which can be dipped into, searched systematically, or mined as a source of further reading. Many Ph.D. theses will no doubt be planned through its calm pages. But really, everyone should be involved in this.

Requirements Engineering
Special Issue of the Cutter IT Journal (formerly American Programmer)
Jim Highsmith, Nancy R Mead, Daniel J Mosley, Balasubramanian Ramesh, Lou Russell, Karl E Wiegers. Introduction by Ed Yourdon.
48 pages, May 2000 (Vol 13, No. 5)
Subscription $485 annually (N. America), $585 (rest of World)
Publisher: Cutter Information Corp (http://www.cutter.com/consortium/)

As **Ed Yourdon** says in his introduction, Requirements Engineering (RE) covers elicitation, analysis, specification, verification and management: a broad scope indeed. This issue of the Cutter IT Journal is devoted entirely to our subject; some earlier issues looked at requirements management tools and techniques.

Nancy R Mead of the SEI writes about Requirements Management and RE, arguing that 'You Can't Have One Without the Other'.

"One of my students once told me that her organization had established practices for requirements management so that they could "check off" the [SW-CMM] requirements management KPA, but they did not develop any software requirements!"

She is rightly scathing about attempts not much more sensible than this to drive a wedge between managing and engineering requirements. The new SW-CMM (version 2.0 Draft [6]) now includes three goals for Requirements Management, namely a repeatable process, an allocated requirements baseline, and consistency of software project plans, activities and work products with the allocated requirements. Hence "it is pretty clear" that the CMM mandates management activities that depend on already-established requirements. These only make sense if good RE practice is also followed.

Mead discusses, with examples, what happens when managers focus on meeting the CMM without regard to software engineering principles, or when engineers focus on the technically interesting parts of RE – such as elicitation techniques, or notations – without keeping track of changes. And she discusses the temptation

"..in industry, when a team goes into crunch mode. Good software engineering practices are often the first thing to go. A team may decide to 'fix up' the requirements at the end of a project, rather than trying to keep them current throughout. To the team, maintaining requirements seems like an overhead task."

This is sobering stuff, and excellent ammunition too, if you are faced with similar situations.

Karl E Wiegers, author of a textbook on Software Requirements, discusses key RE practices in an article entitled 'When Telepathy Won't Do'. He argues that

"Some highly exploratory or innovative projects can tolerate the extensive rework that results from informal requirements engineering. Most development efforts will benefit from a more deliberate and structured approach, though."

This is undoubtedly close to the truth for large and critical systems, but it runs against the grain of the current trend towards rapid development (as exemplified by Kent Beck's Extreme Programming, among other approaches). It depends on your audience and the scale of their applications.

Wiegers goes on to explain the traditional distinctions between business, user, and functional (system) requirements, before presenting his own view of the requirements development process and his taxonomy of RE. He then concentrates on key practices for the stages he identifies – elicitation, analysis, specification and verification. It is sensible and practical stuff, aimed at large projects in large organizations.

Balasubramanian Ramesh looks at Implementing Requirements Traceability. He explains in simple terms what this central and much misunderstood concept of RE actually means. To those of us who deal with traces every working day, it is hard to remember that the concept is less than obvious. Ramesh distinguishes low- and high-end users of traceability. Low-end users

"use simple traceability schemes to model dependencies .. allocation .. and compliance. [They] do not capture process-related traceability information"

whereas high-end users

"employ much richer traceability schemes, thereby enabling more precise reasoning about traces"

Not surprisingly, the low-end users tend to sacrifice traces when a funding crunch comes along, whereas the high-end users think that "may prove to be a poor choice".

In other words, the cost/benefit argument about traces is much the same as that about writing requirements in general – it takes some sophistication to see the benefit and more to realize it on projects.

Ramesh points out that high-end users prefer to use tools that are embedded in their development environment, but that many requirements tools – he names DOORS as his example – address only limited aspects of the system development lifecycle. He goes on to argue that incompatibility among CASE tools is "particularly detrimental to the maintenance of a dynamic traceability practice, and points out the need for semantic (logical) links to be maintained across artifacts maintained by different tools. While agreeing with his intentions, I'd like to reply that platforms such as DOORS already provide for traceability across interfaces to a wide range of other tools; many of these interfaces are automatic, allowing developers to move seamlessly between specifications, design, and tests held in different tools. On the other hand, Ramesh's emphasis on traceability documents as 'living' entities seems entirely right.

Carol Dekkers and **Maurizio Aguiar**, in Double Duty Metrics, discuss how to use functional sizing to gauge requirements completeness.

Function Point Analysis (FPA) has traditionally been used to estimate software development cost. The authors argue here that the technique can equally be used to uncover requirements defects and omissions.

After pointing out that requirements "can single-handedly bring a software project to its knees" and the difficulty of judging whether a 600 page formal DoD SRS is complete, they argue that neither abstract frames of reference like structured analysis, nor informal personal experience are adequate to guarantee good results. Instead, they suggest, the formality of FPA offers a reliable route to quality. The function point approach identifies the required functions and the system scope, creates a data or entity-relationship model, counts the transactions and measures the complexity of user constraints.

Problems such as that an entity is defined but never used in a transaction, or is used but never defined, are highlighted during the process. Simple patterns such as 'AUDIO' – entities typically have Add, Update, Delete, Inquiry and Output operations – allow possibly missing operations in the specifications to be highlighted.

All this seems sensible and indeed highly beneficial as far as it goes. Where readers may feel doubtful is whether the rather traditional functions-and-dataflows approach is right for them. Not all systems are transactional, and much development is object-oriented; the applicability of FPA in such cases is uncertain. The claim that FPA is 'objective' is also disputable; all assignments of reality to a set functions are to some extent arbitrary.

Lou Russell, in Disaster Looks Good to Me, looks at the disaster waiting to happen when nontechnical project managers, to save time or face, approve requirements without validating them. Then she describes in plain and practical terms what to do.

The article provides good and useful 'checklists for success': a project scope, risks to be faced, constraints, business objectives, system (project) objectives, process needs, data/information needs, and validation. Each is properly documented in the article with an owner, reviewer, and 'Risk if skipped' – in other words, project managers, here is what will happen if you don't do this bit.

Russell manages to cover some very plain and straightforward pieces of advice in a fresh and readable style. She is obviously aiming primarily at the typical business data processing project, but her principles are more general than that. Highly recommended.

Daniel J Mosley discusses how to define test requirements for automated software testing. Perhaps one day it will be obvious to everybody that software should be tested against its requirements, or to put it the other way around, that requirements are incomplete if we can't see exactly how they are to be verified. But as yet it is not.

Mosley makes the obvious and correct points that to verify you must specify, and that the link between the two should be automated. However this seems to stem from a specification of test requirements, to be linked to test conditions; he doesn't discuss expressing the system requirements as scenarios/use cases, and then automatically generating test cases from those, which is a large task in test specification. He mentions favourably Rational Software Inc.'s products several times, and goes so far as to feel obliged to state in a side-note that he is not affiliated with them in any way!

The article ends by describing templates for test requirement specifications, and then illustrates in some detail what the test cases should look like, and argues that these should be automated and linked to the test requirements. These are good points but possibly somewhat obscured by his choice of terminology.

If this sort of publication helps the message of requirements engineering to reach a wider audience, especially of business executives, then it is very much to be welcomed. For busy senior managers, these few pages of concentrated knowledge may be the ideal introduction to the advantage that businesses can gain through proper handling of requirements.

Readers may also be interested to note that Cutter Information Corp. issues a free newsletter, the Cutter Edge (www.cutter.com/consortium), on IT issues affecting business.

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. The CREWS project has developed two prototypical tool suites which can be employed, e.g., as extensions to the Use Case approach in object-oriented systems engineering. One shows traceable multimedia-based current-state analysis and animation of future scenarios, the other provides guidance for the creation and analysis of text scenarios and for the systematic generation of exception 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. Special 2001 Personal Rate £45.00 (A saving of £19.00).

Mailing lists

The SRE list

The SRE mailing list aims to act as a forum for exchange of ideas among the requirements engineering researchers and practitioners. To subscribe to SRE mailing list, send e-mail to listproc@jrcase.mq.edu.au with the following line as the first and only line in the body of the message:

subscribe SRE *your-first-name your-second-name*.

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-Actors

The committee of RESG

Patron: Michael Jackson,

Chair: Bashar Nuseibeh, Computing Department, Faculty of Maths and Computing, The Open University, Milton Keynes, MK7 6AA, UK. E-Mail: B.A.Nuseibeh@open.ac.uk, Tel: 01908-655185, Fax: 01908-652140

Treasurer: Neil Maiden, Centre for HCI Design, City University, Northampton Square, London EC1V OHB, UK. E-Mail: N.A.M.Maiden@city.ac.uk, Tel: 0207-477-8412, Fax: 0207-477-8859

Secretary: Wolfgang Emmerich, University College London, Department of Computer Science, Gower Street, London WC1E 6BT, UK. E-Mail: W.Emmerich@cs.ucl.ac.uk, Tel: 0207 504 4413, Fax: +44 171 387 1397

RE-Creations

To contribute to RQ please send contributions to Pete Sawyer (sawyer@comp.lancs.ac.uk). Submissions must be in electronic form, preferably as plain ASCII text or rtf.

Deadline for next issue: 25th May 2001.