BCS™

Requirenautics Quarterly

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

©2005, BCS RESG http://www.resg.org.uk Issue 35 (March 2005)

Contents

RE-Soundings

From the Editor

RQ is pleased to offer a rich selection of papers and reports of varied kinds in this issue.

There's a new section in the shape of some *RE-verberations*, echoing news items that RQ hopes will be of interest. RQ would specially like to hear from readers whether this is a feature worth continuing.

A very welcome piece of news to any editor's ear is that readers from both industry and academia have contributed their opinions. RQ is also happy to continue the RESG's tradition of collaboration with other institutions, in this case the IEE, which held a very successful event on UML for Systems Engineering.

Coming up very soon now is a first for the RESG, a major programme of linked events on the theme of the i* modelling language. If you've ever felt that the analysis and modelling approaches that you've been sold have rather little to say on the subject of what people really want – their intentions and goals of different kinds – then i* may be for you. Where both conventional and object-oriented approaches tend to dive into design (with functions or classes), i* focuses on the human roles, their many dependencies on each other, and their 'soft goals' such as to achieve higher performance. Is i* requirements engineering's best-kept secret? Maybe April is the time for you to find out.

Ian Alexander,
Scenario Plus

Chairman´s Message

As I write this we're already well into 2005. The deadline for RE'05 paper submissions has passed and a number of other RE-related events are already gearing up. After a bit of a hiatus, the next RESG event will be the i* event on the 20th - 22nd April.

Following that, we have scheduled an 'audience with' event some time in May (watch the website) and a tool vendors' challenge in July. That should tide us over nicely for the run-up to RE and then we'll be into our autumn events again. We hope that one of the events just over the horizon will be a medical informatics seminar, replacing the one we'd scheduled for this January but had to be cancelled.

Right now, though, it's Easter-ish and the academics amongst us will have finished delivering their bachelors and masters-level RE courses and will probably be awaiting the external examiner's feedback on their exam questions (unless they've been semesterised). Since RE-specific modules have become mainstream in software engineering and computer science degree programmes in recent years, I often wonder if or how employers perceive this addition to graduates' skill-sets. I guess many employers simply aren't aware of them because fresh graduates aren't always let loose on clients or customers.

I was lucky because that wasn't how it was in my first computing job. I got to meet real stakeholders and real users occasionally. I think they felt that they had to use the short time we had to get over their required functionality, while I didn't understand enough about elicitation to find out much about what they did or why they did it. I received a valuable lesson, though, when I was sent to watch an exercise on Salisbury Plain. I realised that our users, who were army signallers and had a very complex analytical job, were sat in the back of a truck in the middle of a maelstrom of rolling tanks, screeching aircraft and very loud bangs. It must have been very hard to concentrate! If nothing else, the event did help me realise why you need to understand the environment in which people perform their jobs. That's something I try to pass on to my RE students. I hope they remember at least some of it when they become requirements engineers, business analysts or product managers.

I hope you enjoy the RESG program for 2005. Please remember that we're open to suggestions for future events.

Pete Sawyer
Computing Department, Lancaster University

RE-Treats

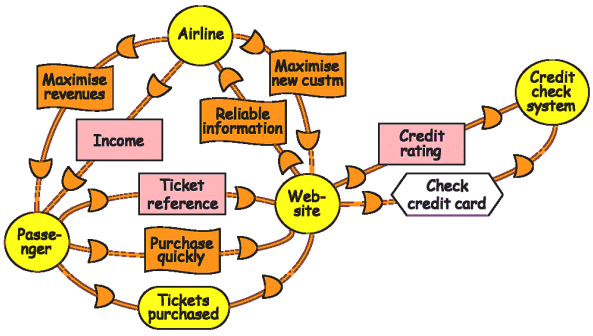
For further details of events, see the RESG website at www.resg.org.uk

Forthcoming events organised by the RESG:

The *i** Conference: Goal modelling with the *i** approach: 3 days of events

20 April 2005, **Tutorial and 4 Talks** (from industrial projects), City University, London

21-22 April 2005, **Invited Speaker Workshop**, University College, London



Example i* Strategic Dependency Diagram
eg Passenger depends on Website to achieve goal of purchasing tickets quickly

Concerned about how to model the goals of diverse actors in your organisation or project?

Unsure how to explore complex goal trade-offs during your requirements process?

This conference contains events for both practitioners and researchers. It will introduce, tutor and investigate the ***i**** approach for modelling and reasoning about the goals of heterogeneous systems.

i* (pronounced eye-star) is a powerful approach for modelling and reasoning about the goals of heterogeneous actors in business and socio-technical systems, and for choosing systems architectures that best meet these goals.

The conference will be in 2 parts.

- On Wednesday 20th April there will be a half-day tutorial on ***i**** followed by 4 presentations of the use of ***i**** on industrial projects.
- On Thursday and Friday 21st and 22nd April there will an invited speaker workshop to investigate and extend the ***i**** approach.

For registration, contact Neil Maiden (N.A.M.Maiden@city.ac.uk).

An Audience with David Parnas

A chance to hear one of the big names in software engineering.

Wednesday 25th May 2005. Imperial College, London.

- 10am: Tutorial.
- 2pm: Seminar and Questions.

See the RESG website (http://www.resg.org.uk) for full details.

Contact: Bashar Nuseibeh (B.Nuseibeh@open.ac.uk)

Introduction to Requirements

1-Day course at the IEE, Savoy Place, London WC2

Thursday 12th May 2005

This proven introductory course, taught by Ian Alexander, covers the systems engineering context of requirements work, requirements elicitation and analysis, writing & reviewing, and requirements management. Participants work through practical group exercises.

A discount is available to RESG and IEE members.

Registration: online form on the IEE website:

http://www.iee.org/events/intro-req.cfm

Tool Vendors Day and AGM

Wednesday 20th July 2005

After the (short) Annual General Meeting, each Vendor will speak on how their tool meets the following challenge:

> "You are acting as requirements management consultant to a client who wants to automate his existing multi-storey car park with time-stamped ticket-issuing machines, payment machines, closed-circuit television cameras to deter both theft and non-payment, and automatic barriers operated by validated (paid-up) tickets.
>
> The client's systems engineer has advised that the requirements should be organised into a list of stakeholders, a set of stakeholder requirements, a system specification, a project dictionary, and a list of references, with traces between these (the dictionary and references both receive traces from all the other documents; the specification traces to the requirements, which trace to the list of stakeholders).
>
> The requirements will certainly need to be prioritised, approved (or rejected), and then have their status tracked through to final acceptance of the automated car park."

Show (without spending time restating the problem) how your tool handles this challenge. Illustrate briefly the steps you would go through to structure the requirements, traces, priorities, and status in your tool.

Present slides on each of the following topics:

- setting up the information structure skeleton;
- importing the requirements from Word or text files;
- setting up the traceability;
- prioritising and approving the requirements;
- tracking the status of the requirements;
- checking the completeness of the traceability;
- any special features of your tool that assist with these tasks.

Registration: contact David Bush (David.Bush@nats.co.uk)

RE-Calls

Recent Calls for Papers and Participation

RE'05: The 13th IEEE International Requirements Engineering Conference

August 29th - September 2nd, 2005, Paris, France

http://www.re05.org

Notifications sent to authors: 22 April 2005
Camera-ready papers received: 03 June 2005
Doctoral symposium submissions: 28 April 2005
Poster and research demonstrations submissions: 28 April 2005

ICSE 2005: 27th International Conference on Software Engineering

15-21 May 2005, St Louis, Missouri, USA

http://www.icse-conferences.org/2005/

SCESM'05

4th International Workshop on Scenarios and State Machines: Models, Algorithms and Tools, 21 May, 2005 Saint-Louis, USA (An ICSE 2005 workshop)

http://www.info.fundp.ac.be/~ybo/scesm05

CaiSE'05

13-17 June 2005, Porto, Portugal

http://www.fe.up.pt/caise2005

REFSQ'05

The Requirements Engineering: Foundation for Software Quality Workshop will be held in connection with CaiSE'05.

ESEC / FSE

10th European Software Engineering Conference / 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering

September 5-9, 2005 - Lisbon, Portugal

http://esecfse05.unl.pt

SPLC-Europe 2005

9th International Software Product Line Conference Rennes (France), 26-29 September 2005

http://www.sse.uni-essen.de/SPLC2005

MoDELS'05

8th International Conference on Model Driven Engineering Languages and Systems (formerly the UML series of conferences) October 2 - 7, 2005, Half Moon Resort, Montego Bay, Jamaica

http://www.modelsconference.org/

ASE 2005

20th IEEE/ACM International Conference on Automated Software Engineering, November 7-11, 2005, Long Beach, California, USA

http://www.ase-conference.org/

RE-Readings

Reviews of recent Requirements Engineering events.

IEE Seminar on UML for Systems Engineering

Savoy Place, London, 17 Feb 2005

Jon Holt (Brass Bullet) chaired the meeting. He said that UML (Unified Modeling Language, if you're from an isolated desert island) did three things — it helped people deal with

- Complexity
- Understanding problems and solutions, and
- Communication.

Bruce Powel Douglass (I-Logix) ("Author of many books, world leader on UML, and Chief Evangelist at I-Logix itself" said Holt) gave the keynote talk on Model-based Systems: SE in the new millennium. His interesting and wide-ranging talk seemed to introduce several commonly-held views of what Systems Engineering (SE) is; he didn't draw attention to these so they are flagged in the following account (in parentheses).

Classical paper style (he showed a lovely photo of a pensive team surrounded by sheets of paper) is labour-intensive. Text is hard to validate — you sit around the table and think. Text is also ambiguous, whereas real-time safety-critical systems are precise. The handoff from paper means manual translation into languages for software, electronics, mechanical design, etc.

Model-Driven Development (MDD) tries to avoid this by making models executable, validatable directly. UML 2.0 and SysML languages are being applied on very large projects like joint strike fighter.

But most projects today still use text, and maybe Excel spreadsheets. "That makes me afraid when I fly in a plane."

As systems become "intelligent", software becomes the distinguishing aspect between system designs and capabilities. Software is special; it is non-physical (no worrying about where to drill the fixing-holes in the racks), so it actually *is* its specification.

MDD at its most basic involves editing models on a desktop, and generating code from models. But then we can simulate, execute, formally test, and debug, feeding corrections back into the models. All that works today. (This is the software engineering view of what SE is.)

UML supports various kinds of modelling:

- requirements modelling — use cases bind requirements into "coherent operational units";
- structural modelling — the parts of a system that fit together
- state behavioural modelling (statecharts)
- algorithmic behavioural modelling (activity diagrams)
- performance modelling — tags and constraints annotate UML models (slight extensions to the language), enabling performance analysis tools

The UML diagrams are views into the semantics of models.

Over 50% of system development cost is in testing and verification (he called it validation). This cost can be reduced by generating "test vectors" automatically from models — the same ones that characterise the requirements. Obviously this is better for software than using text on paper (even if we have reservations about doing the same for hardware).

MDD can fail. E.g. "Napkin designs", quick informal sketches of designs that can't be executed, validated, or understood precisely, can lead to trouble on projects.

SysML is a general-purpose modelling language for Systems Engineering applications; it supports the entire life-cycle (not just for software). It builds on a subset of UML 2.0 with minor extensions (maybe 10%). OMG SysML is now at 0.9 so will reach final release version 1.0 this year. The extensions are only where vital, so the standards can "co-reside". New diagrams include Parametric and Requirement models. Activity diagrams are extended to allow continuous (not discrete) behaviour. E.g. you can model analogue quantities such as angular velocity and thus predict how systems behave. This is far beyond the simple flowchart-step behaviour of standard UML activity charts.

SysML Class diagrams can depict mechanical, electronic, software, and mixed objects (yes, we thought you could do that already). So you can show how an assembly satisfies requirements (shown in little

boxes inside boxes) with visible traces; rationales can be attached to those traces. SysML Parametric models can show physical laws (Newton's first law, whatever) and you can connect the boxes together "so you can do systems engineering in UML". (That is the control engineering view of what SE is.)

Requirements traceability relationships covered include *satisfy, verify, derive, trace*. A use case bubble is linked to textual requirements with "*qualify*". The result is much more verifiable than a textual model (though perhaps not much more than a structured text in a conventional requirements traceability tool. Certainly the overlap between requirements analysis tools like I-Logix Rhapsody and requirements database tools like Telelogic DOORS is becoming large).

The vision is of a wholly model-driven SE approach. A spiral life-cycle, with models from the start, helps to eliminate nasty surprises (this is certainly the systems integration view of what SE is). The "Increment Review" on his spiral diagram is labelled "(Party)" — hey, we're making progress.

Bruce Powel Douglass' books include *Doing Hard Time*, *Real-Time Design Patterns*, and *Real Time UML*, all in the popular Booch-Rumbaugh-Jacobson UML series from Addison-Wesley.

Ljerka Beus-Dukic asked how he related use cases to parametric models, given that use cases were poor for non-functional requirements. He disagreed; a use case was a bag of requirements; if a requirement was a verb ('do that') phrase, then a parameter was an adverb ('do x quickly'). That could perfectly well be placed inside a use case. Then it could be shown to be met by tracing from a parametric model, going down several levels of detail if need be.

Paul McNeillis (BSI) spoke on Cognitive Mapping and UML Modelling: comparing book and mind. The BSI produces about 1800 standards/year (quite a rate!).

A cognitive map is a visual representation of the ways individuals view a situation. Boxed texts are linked by causative ("is likely to cause") arrows; there is little grammar but models can become large. The aim is to be accurate, to represent what is in an individual's mind. It's for messy, complex, qualitative problems such as strategy. You don't solve such problems; you navigate through them.

Why bother? Organisational learning may be the only sustainable source of competitive advantage. Cognitive maps can address things like shared understanding within an organisation (whereas most approaches can't). Problems include ambiguity, equivocality (meanings are related to attitudes) and crypticality (one person's meanings are obscure to others). Eg "These procedures provide additional guidance on safe working practice requirements…" has a familiar ring: it is bureaucratic and hard to interpret. Does it mean that the guidance is mandatory because it interprets the requirements, or optional because it is separate from the requirements? A pair of UML class/subclass diagrams can represent the alternative meanings unambiguously, and people can then discuss them openly. So, there may be value in using UML alongside cognitive mapping.

UML is a highly-developed analytic visual grammar; cognitive mapping is a simple holistic way of showing weakly-structured, even tacit concepts explicitly. The "book" may document organisational procedures, standards, and regulations — what an organisation claims that it knows, claims that it does, in the boardroom. The "mind" may contain tacit operational interpretations of these, and must deal with what really happens on the shop floor. What happens when we compare these? Not surprisingly, book and mind are often poorly aligned; what management think is happening is often far from the truth. So, creating cognitive maps might help businesses run a little better.

Commander **Bill Biggs** RN (MoD Integration Authority, Abbey Wood) spoke lucidly on Modelling MoDAF – the Defence Systems Architecture Framework for the MoD. Improvisation fails in large systems; things need to fit together, which demands planning within an agreed framework.

Network Enabled Capability (NEC —in the USA, this is called net-centric warfare) is the linking of sensors, decision makers and weapon systems to achieve victory. This creates a single large long-lived system with many components that must fit together. NEC is a priority; it makes a framework like MoDAF essential.

Smart Acquisition identifies required capability, translates that into user-centred requirements, and passes those into integrated project teams —with the danger that they become stovepiped. MoDAF is meant to avoid that.

The House of Commons Defence Committee has roundly criticised waste and inefficiency at the Defence Procurement Agency. MoDAF is part of the answer.

MoDAF includes an enterprise architecture, supported by a metamodel and an object taxonomy (both of which can be modelled in UML). The MoDAF itself is essentially a style guide for building architectures.

MoDAF is built on DoDAF; other frameworks like Zachman's were looked at. DoDAF had to be tailored for the UK context, ie the MoD lifecycle (moving away from CADMID[1]) and processes. MoDAF includes operational, system, and technical views.

A repository allows architectures to be co-ordinated. 20 of 33 views will be in UML. XMI will be used for tool interchange. Most of the tool support will be via UML. The metamodel extends SysML. It's crucial to be able to interchange tools, to represent taxonomies

[1] Concept – Assessment – Demonstration – Manufacture – In-Service – Disposal, a 'Smart Procurement' defence systems engineering life-cycle

(there are many in the MoD), and to have vendor support — happily, several tools already claim to be MoDAF-compliant even though its developers don't yet have anything to comply with! A validated framework should be ready by June 2005; feedback is welcomed. Find out more from the www.MODAF.com website.

Richard Barrow (Railway Safety & Standards Board) spoke on Setting New Boundaries – Applying UML to Railway Standards. There are many boundaries in the UK railway. After 25 years of being a signals engineer Barrow joined the RSSB; he'd never heard of UML before, isn't a systems engineer, and has no idea of formal methods. How could UML help with standards? How much of it? How much skill was needed? Would the benefits justify the cost?

RSSB asked Jon Holt's Brass Bullet to translate one standard, Train Activated Warning Systems, into UML. The aim was to identify inconsistencies in the standard, suggest how it could be restructured, and help the RSB to identify the benefits of UML. The reply was that the standard was so bad that it couldn't be expressed in UML!

So, after 2 days' UML training at the IEE (from Holt), Barrow modelled what a Standard was, as a class diagram. This was easy. But, describing how a system is built isn't what the RSSB needed; what was wanted was a model of its purpose. So he tried Use Case diagrams instead. The stakeholder roles were interesting, but the Darling review of UK railways put the project on ice. However, having the domain knowledge in UML helped Barrow to think clearly, without having the UML tail wagging the dog.

A separate venture is to take ERTMS (already available as products) and define its scenarios. Modelling the whole railway in class diagrams is a huge task! But modelling the behaviour, what the railway does, what its people do, what their goals are, is more tractable. So again, Barrow resorted to use case diagrams to describe people and their (functional) goals. This lets the RSSB describe complete sets of activities (in a set of diagrams).

Then he listed all the stakeholders in a class diagram, grouped into operations people, managers who don't touch the system, etc. This separated people whose roles would change with the railway's equipment, and those whose roles wouldn't. He did the same for the equipment. The class diagrams made it easy to draw the scope of the new system as a boundary line. Every part of the railway is modelled as a component, a boundary, an actor, and a class —so he found he had an RSSB-specific UML Profile. He had reached a consistent level of detail, and a rationale for the model. It should now be possible to (re)model existing standards.

UML is extremely useful for modelling documentation such as standards. If you are clever enough to understand what you want to model, you are certainly clever enough to model it in UML. UML allowed Barrow to structure his thoughts, to put them down on paper.

UML might also be helpful in other areas, such as modelling incident inquiries.

Ian Alexander asked if Barrow believed that all rail standards should be modelled in UML. Yes, said Barrow. Standards were about risk, and models let you see what the risks were. We didn't have signalling principles today, we had equipment-specific principles expressed as text. If we got the standards right it would make a lot of things easier to develop.

Mike Brownsword (Lloyds Register) and **Rossi Setchi** (Cardiff University) spoke on Picturing Risk: Connecting Industries. Factors control Risks; they also cause them. So, models of Factors and Risks can help. Risks can be subtyped into Business Risks, eg Financial ones, and Technical Risks, such as safety. Factors can be subtyped into Standard, Lifecycle, Business, and Competence.

EN50126 is a safety standard to do with life-cycles (in 14 phases); Barry Boehm's Spiral model is also of a life-cycle. But the phases do not match! And, we don't do projects as neat V-models. The Boehm spiral can be remodelled in UML (as a cycle of 4 stage-boxes linked in a circle by arrows, with 'In service' as the final box off to the side). That makes it possible to compare with EN50126.

BS 6079 on Risk Management has a task with no name! Unfortunately it's linked to all the other tasks. It includes databases and "communication and English". Obviously there is a weakness in the formerly implicit model underlying the standard.

It can take many months to arrive at a simple UML model of a business ("Resource carries out Technical Assurance or Work; Risk is reduced by Technical Assurance" etc, modelled as classes with arrowed relationships).

Risk can be modelled (using UML) in various ways and at various levels. Four models were illustrated. The Safety models look at 'hazards' or 'harm'; other models look at 'return' or 'effect'. Outcomes can thus be subtyped as positive or negative. All four models deal with frequency.

All the risks can be fitted into a single model —risks of different kinds interact, as there are trade-offs between business efficiency and safety.

Defining "risk" in UML enables people to be clearer about which meaning of this much-used term is intended.

Michael Emes, **Doug Cowper** and **Alan Smith** (UCL) spoke on Using UML to model business processes: a case study based on instrumental use and procurement. They used 3 UML model types: class, statechart, and activity diagrams.

Doug Cowper described making class models of Stakeholders; of the business, with goals, processes,

rules, plans, employees and the business itself; and of the effect of instrumentation on the business' bottom line. This involved modelling the use of instruments in the business (to help chemists make drugs, actually). A class/subclass hierarchy described the types of instrument.

He then moved on to modelling more abstract things, such as instrument procurement, and the supply chain strategy. Something that sounds simple but was quite tricky was how to model employees.

- You could use just one class, Employee, and place UML rolenames on the links (UML associations) to other classes, eg Employee—(operator)—Instrument.
- You could use actual job titles, eg Engineer, as class names.
- Or you could create abstract or logical "hypothetical" classes such as Operator, which could be filled by both Engineers and Physicists, according to need.

This last was chosen, as it is more reusable and tolerates changes in job descriptions and skills; it also permits business process reengineering on employees' job scopes, making it possible to improve them.

The output is a static UML structure but it is developed in a mixture of ways: top-down for stakeholders; bottom-up for instruments and processes, ie looking at what the business currently did; linking the two with technology management models (eg of procurement). The work led to the proposal of a new procurement process. The activity diagrams lead to simple process charts that the organisation can use to implement each process.

Matthew Hause (Artisan) spoke on SysML — Making UML more effective for systems engineers. He worked in aerospace for many years. Artisan now has many aerospace customers including BAES, Lockheed Martin, Westland Agusta etc. He's a member of OMG and the SysML team.

UML is the de facto standard in software engineering (the other "SE", or SwE as systems people sometimes say). UML 2.0 offers "many" features for systems engineering.

SysML comes from industry, and not just military aerospace: while BAES, EADS, Thales, Northrop Grumman etc are there, so is Boeing, and so is Deere (tractors to you). Vendors are represented but are not dominant; there is no Microsoft calling the shots. Words like "component" which sound Microsoftish are eschewed in favour of "assembly" which sounds neutral.

SysML includes the Assembly Diagram (systems of systems of …) with the usage guidelines for Composite Structure, using an "assembly" (enclosing class) and "ports" (subclasses of "parts", apparently) for interfaces (rather than the rather nice ball-and-socket notation of UML 2.0). Connectors can be nested, too. The Activity diagram permits several new options. Allocation (of requirements to components, etc) is supported. Information flow is extended to support parametric modelling. Class notation supports dependency sets and specialisation hierarchies. You can specify views and viewpoints. The collaboration diagram is now called the communication diagram, but SysML doesn't use it. For modelling continuous systems, things can happen in parallel in real time. And so on.

Parts are properties enclosed by assemblies and typed by classes. The assembly diagram is nice because using class diagrams with lots of aggregation relationships quickly becomes unreadable. But draw a diagram of a box with ports connected to another box with ports, all inside a big system box with its externally-available ports, is instantly understandable. So the rule for good SysML is, find a way to make your diagrams look simple and obvious — they'll probably then be right.

Requirements are more than use cases; NFRs and environmental constraints need to be defined and then linked to model elements. Pure requirements are modelled in groups (in Packages). Traceability and impact analysis can all then be done in the same tool environment.

Profiles are just stereotyped packages containing model elements customized for a specific domain such as automotive. Customisation may be via UML stereotypes, tagged definitions, or constraints. The automotive domain contains an Environment, which includes weather, terrain, GPS satellites, and so on. These things are UML model elements; but they can be associated with a graphic means of presentation suitable for board members (the navy calls this the "Admiral" view), eg "weather" appears as a little cloud shedding raindrops, "satellite" as a little cube with solar panel wings, and so on. It's still a model but it's easier for children, chairmen and admirals to understand.

If 90% of UML is for software people, 10% is for systems people. SysML is for them. So, it's specialised away from software. For instance, it has to model the context (people, environment) for systems to be understandable. You have to model people-people interactions — that's most of the work in business process analysis.

Jon Chard (Telelogic) was scheduled to speak on MDA and Systems Engineering but was taken ill during the day.

Therefore, **Jon Holt** spoke instead on applying SE to procurement management. He managed to be both fresh and informative, at short notice.

To managers, "an engineer" means spanners and oily rags, he said. Luckily the verb "to engineer" means to apply cunning and ingenuity, so some respect remains. Systems Engineering means applying common sense.

The evils of life include communication problems (between english and american, say); projects have failed because "to table" means to make a motion active in England, but to shelve in America. Clinical psychologists use "to section" to mean to commit to a madhouse; midwives use it to mean to deliver a baby by a Caesarean operation. Miscommunication and misunderstanding can never be banished, but modelling can sometimes keep them at bay.

UML 2.0 should have got rid of the horrible stickman. Even the term actor implies people. Jacobson should have chosen the other meaning of Swedish's borrowed word aktör, which is role. Stakeholder roles can readily be modelled as UML classes and subclasses.

UML offers 4 diagrams for scenarios — sequence, activity, communication and timing. A scenario is an instance of a use case (a possible path, to be precise) to meet a particular requirement. A sequence diagram can be read (turn your head through 90°) as a Gantt chart. So, if the requirements are project goals and the processes are project activities, the sequence diagram is the project schedule. You can use the model to drive the schedule and you'll get realistic answers. That might be uncomfortable if you prefer soft soap and mean to switch jobs before the project ends.

So, modelling can be applied to procurement processes such as proposal preparation and tender evaluation. The output doesn't need to look like a UML model, but the model does build understanding.

The speakers formed **a panel for the closing session**.

Were there plans for a query language? The OMG is looking at SQL and OCL (for constraints). The problem is that UML has so many uses that tools have to be very general to be understandable, or must be limited to specific domains.

Did code generation create efficient code? There are possible mappings from many models to code; but there's no point making models which just reflect code —you might as well write the code at once. It also depends whether you're using UML for simulation, or napkin models, or for actual production-quality code. The latter is obviously most problematic. Tool vendors distinguish themselves by what they do with models and code; they offer widely differing features.

Does UML just help you check your specifications? Bruce Powel Douglass said that if you knew that your specifications were complete and consistent, that would be a huge help. Matthew Hause said that finding out what code did, reverse engineering a model of what it did, was often enormously valuable.

Janos Korn said that block diagrams could be read either symbolically or in natural language. But UML used natural language represented pictorially. Wasn't there a large amount of uncertainty about what some of the more abstract diagrams meant? These diagrams had no facility to accommodate meaning. They needed theoretical backup to support the mapping between models and the world. Simple statements in natural language (like "John likes Mary") can be represented in predicate calculus.

Bruce Powel Douglass replied that there was a formal or semi-formal meaning and so UML diagrams could be used for reasoning, and some people use them like that. The combination of formal and sem-formal (as for use cases) semantics is useful, supporting simulation, proof, and human reasoning where appropriate.

Matthew Hause said that a software engineer knows that a class is a picture of the software object he just wrote. A systems engineer has a different view. You need to explain the context and intent of the diagram to understand what it's saying. Several research groups are looking into marrying VDM and other formal specification languages with UML to provide formal semantics when needed.

It was a full and very stimulating day, with a rich variety of talks, in a comfortable environment (expertly maintained by the unobtrusive technical assistance, catering staff, and administrative team).

RE-Papers

Requirements are a Project Management Tool

Suzanne Robertson,
The Atlantic Systems Guild Ltd

suzanne@systemsguild.com

Suzanne Robertson is a principal and founder of The Atlantic Systems Guild and joint originator of the Volere requirements techniques. She specialises in helping organisations to set up relevant requirements processes and build their own knowledge models, does requirements audits and teaches organisation how to write and audit their requirements.

Listening to music, travelling to work, making a cup of tea – these are familiar activities. In fact they are so familiar that we can carry them out automatically without having to decide how to do them. Our experiences have helped us to develop inbuilt patterns that we use over and over again. Now suppose you are listening to Rachmaninoff and the music suddenly becomes much too loud; or there are road works blocking your usual route to work; or there isn't enough assam tea left in the tin to make your pot of tea. What do you do when the reality does not fit your pattern? You make some adjustments to tune the situation. You turn the volume down; you stop and have a look at a map to find an alternative route, you decide to risk mixing the remaining assam tea with

some orange pekoe. We know how to tune situations in our everyday lives because we have patterns containing consistent variables that we understand and know how to adjust to. When managing a project we do something very similar.

An experienced project manager monitors the project's actual behaviour against the expected pattern. When there is a variation then the manager tunes the project by observing one or more variables, analysing them and making adjustments to things like workload, task assignment, version content. The most useful variables for tuning a project are those that are:

- identifiable early in the project
- traceable throughout the life of the product
- reflect the real work that has been done

This is where requirements come in.

I am starting with the premise that your project is using a disciplined, consistent approach to defining requirements, both at summary and atomic level. Provided that this is true, then the project manager can use the requirements themselves as a tuning and decision making tool.

Early in your project

The requirements engineers do their first iteration by skimming across the top of the whole project. They are looking around at project's landscape and identifying the boundaries and necessary depth of the investigation. This provides a number of requirements-related deliverables that the project manager can use:

Requirements Deliverable	Measurables	Decision Making Input
Stakeholder Analysis	Number of stakeholders, roles and knowledge expectations	Input to sociological complexity estimate and knowledge gap analysis
Project Goals	Purpose and expected Business Advantage of doing the project and Measure of target state	Input to value analysis, prioritisation decisions and change management choices.
Investigation Scope	Number of inputs and outputs bounding the investigation	Input to first-cut estimate of functional complexity
Business Event List	Number of bounded functional chunks	Input to task design and allocation, to risk analysis and to estimate of time/resources needed for discovering requirements for each event
Product Scope – ability to do this depends on whether the product constraints have been defined	First cut estimate of the number of interfaces between the planned product and users and other products	Input to planning early prototypes and simulations

With these quantified starting points a project manager has objective input for recognising changes and making change management decisions. For example, if you started with 30 business events and your team discovers 3 more then you need to adjust the project plan to cope with the 3 extra chunks of work.

During the iterations

Each of the above deliverables provides the starting point for doing iterative detailed requirements discovery and definition. As your team progresses into defining atomic requirements of various types (functional, non-functional, constraint and technological) the results of their work provide more input to your management decisions. Provided you have an agreed definition for what you mean by atomic requirement (this points back to the earlier point about consistency) then the project manager can use the deliverables to make project steering decisions without needing to ask endless questions about progress.

A suggested set of attributes for an Atomic Requirement is:

- Unique Identifier
- Requirement Type (Functional, Non-Functional, Constraint, Technological
- Connection to one or more Product Use Cases
- Description
- Rationale
- Source
- Fit Criterion (precise measure of the meaning of the requirement, used to quantify the requirement, as input to writing tests and to negotiating solutions)
- Customer Satisfaction and Customer Dissatisfaction (input to prioritisation decisions)
- Dependencies with other requirements
- Conflicts with other requirements
- History (creation, review status and any other indicators that you use in tracking where the requirement is in your own process)

- Supporting Materials

By looking at the state of the atomic requirements you can get answers to questions that help you to make steering decisions. For example you can ask specific questions like:

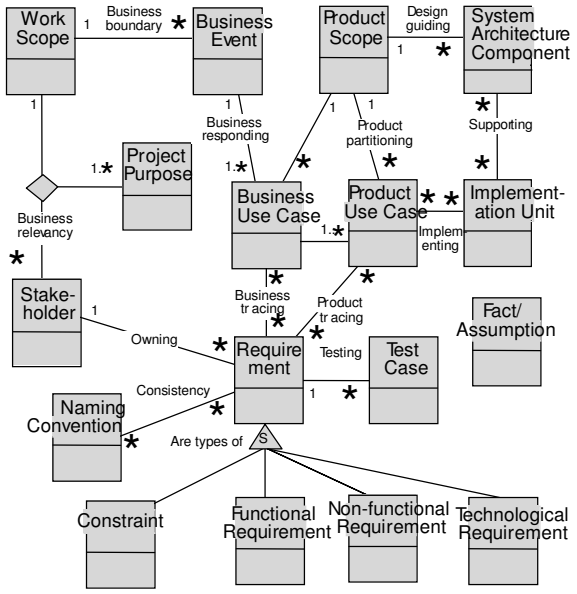
How many atomic requirements have been defined? How many of those requirements have precise measurements and have been though your business review process? How long did it take to get to this stage? Are there functionally related requirements (connected to the same product use case) that could be taken further towards implementation whilst other groups are still waiting for answers to requirements questions?

You can answer all these questions – and more – by looking at the requirements deliverables that your team has produced.

Of course you do not want to do this "looking" manually, and that's where automated tools do the donkey work.

Traceability after the honeymoon

To achieve lifetime traceability you need a formally defined model of the knowledge that you are intending to trace. You can think of this as your requirements for managing your requirements. It serves as a specification for what you would like your requirements management tool to keep track of. This model should reflect the life of a requirement from the time it is discovered to the time that it is implemented in a released product. Hence your knowledge model needs input from all the stakeholders (business analysts, systems analysts, systems architects, designers, programmers, testers….to name a few) who are responsible for any of the transformations or packaging of the requirements.



This is an example of a generic knowledge model that is based on experience of many different projects. You can use this as the starting point for building your own model. Your own model will contain additional classes and associations especially in the marketing and implementation related parts of the model.

Once you have a model that reflects your traceability intentions, then you need the discipline – practised by the whole team – of maintaining the requirements knowledge as defined by your knowledge model. If your requirements are defined in this way then you have some major advantages: you have the basis for making decisions that are supported by precise measurements of the work that has been done, and the team has a model that guides their work by defining deliverables. Hence they are not enveloped in a procedural straightjacket and are able to respond to changes.

Resources

Robertson, James, & Suzanne Robertson. *Requirements-led Project Management: Discovering David's Slingshot.* Addison Wesley, 2005. Contains a requirements knowledge model that defines specific inputs to project management decision making.

Robertson, James, & Suzanne Robertson. *Volere Requirements Specification Template.* 1995-2005. Use this as a checklist for helping you to audit requirements specifications. http://www.volere.co.uk

Robertson, Suzanne *Stakeholders, Goals, Scope: The Foundation for Requirements and Business Models*. This article discusses how to produce early requirements deliverables.
http://www.volere.co.uk/articles.htm

Completeness in Requirements

Venkatesh Vinayakarao
HCL Technologies, India

"Software development is best done iteratively. Requirements Engineering advocates completeness. Does it make sense in attempting to achieve completeness in an iterative project? My client will never tell me that the requirements are complete!"

…Have you heard such arguments? Read on!

There has been a lot of theory, even empirical results indicating the benefits of Requirements Engineering (RE). Still, after several years of propagation, there is a lack of management support and industry adoption of RE principles and practices. The overhead of cultural and behavioral changes required to address RE issues is not considered worth the effort in most project teams. In the real world, managerial skills are still believed to be more than effective enough to drive RE efforts. Some people believe that software can be developed with caution such that the cost of change is always affordable.

Achieving completeness has been considered to be one of the core objectives behind RE. There are some fundamental problems in understanding this concept.

First of all, absolute completeness (of 100%) is practically impossible to demonstrate. If a specification is 100% complete, it will be nothing other than the final system itself. The goal is to verify if the specification is sufficiently complete to proceed with the next task for which it is written; transformation from design to code, or requirements to design, for example. This applies to iterative development also. Fewer requirements do not mean insufficient requirements in order to complete iteration. Assume that there are certain error conditions that must be handled according to the specification; but whose handlers, the analyst missed to capture. Anything not specified is usually left to the choice of downstream developers. The issue arises if the downstream developers either do not know how to address this gap or make wrong assumptions. If the iteration cannot be claimed as complete without knowing how the system should behave under such critical exception conditions, we have a case of incomplete requirements.

Identifying such incompleteness happens usually when the developer finds difficulty in converting the design to code. This is too late in the life cycle and leads to managerial concerns. When the specification grows large, manually addressing such incompleteness verification tasks becomes daunting. Teams start looking for tools that could help them. Tools can only help if the specification is structured, ie it is formal to a sufficient extent for machine interpretation. Much research is being done to understand the nature of formalisms that could help identify incompleteness. State Machine research is an example. The fundamental lesson is that up-front choice of a formal method could save effort, time and money.

Of course, there is no silver bullet. We need to know the right amount of formalism for our project at the right time.

Searching for the Keys to Stability under the Wrong Streetlight

Once upon a time I saw someone crawling about under a streetlamp on a dark night. 'What are you doing?', I asked. 'Have you lost something?' 'My keys.' I knelt down nearby and searched for some minutes without success. 'Are you sure they're round here?' I asked. 'No', said the searcher, 'but this is the only place where it's light enough to look.'

Dr. Islam A M El-Maddah, Ain Shams University,
Faculty of Engineering, Cairo, Egypt

Islam_elmaddah@yahoo.co.uk
www.dcs.kcl.ac.uk/pg/elmaddah

Software stability plays a key role in long-term projects. Clients cannot tolerate paying for software applications that cannot easily be modified and extended.

Failure to arrive at stable software was habitually regarded as the implementer's own fault. Nowadays, there is a strong belief that the keys to stability are hidden on Requirements Street.

One such belief about stability proposes splitting software applications into changeable and unchangeable constructs. This separation is be considered at requirements analysis level, where concepts supporting these constructs are to be considered and modelled. In one way or another, this separation is tightly coupled to the idea of knowledge normalisation.

Have you ever considered refurbishing your house? How much effort and money would you estimate for accomplishing this task? Is it acceptable to pay more money for the refurbishment than for buying a new house that is the same as yours?

What about your car: have you thought about renewing its seats? You would hardly agree to change the seats if that entailed changing the car frame or the inner body! Odd isn't it? That would be unacceptable in most disciplines.

A valuable product should have its own investment programme. Some of its parts and optional sections are planned to be replaced with limited effort that should not exceed the initial price. What makes us wonder is that why these simple ideas are uncommon within the software world.

What is distinct about software applications that devour huge sums of money for maintenance? Is this because many software projects are one-off products or perhaps lack a standardised development process? Why do software firms not take into consideration delivering the software product in a similar fashion to other engineering disciplines?

Many questions need satisfactory answers before software can enter the golden era so long promised, when development efforts will be saved and reused repeatedly, when software products are designed for easy change and adaptation. Some of these questions are:

- What makes cars and houses stable [3] against clients' ever-changing needs?
- Why are other engineering products more stable than software systems?
- Is there a missing effective product-line behind the software product compared to other engineered product?
- Why do people not devote adequate effort to software development methodology and processes compared to other disciplines?
- What makes a software product stable over some period of time?
- Does the client demand certain quality of the required software application?
- Is there an easy way to measure the quality of the developed application?

- What can be changed and what will remain unchanged over time in a software application?
- Is it possible to separate these two parts into different categories?
- How much will it cost to develop software products in this new fashion?

Before answering all these difficult questions, let's see how we can describe stability of thoughts and concepts. Many people may agree with me that replacing one ritual by another depends to a large extent on the existence of a supporting concept or belief behind the ritual. A suitable belief might be that the effort required with the new ritual will be less.

For example some people believe in Modularity: separating artefacts into major concepts and practised ideas.

The ideas can easily be changed or replaced by others once they have supporting concepts; whereas the concepts remain unchanged and facilitate changing one or more of the supported ideas. Thus, for a person to be flexible (having the ability to change interacting thoughts) he/she has to differentiate between concepts that should be fixed over time, and those that must be allowed to be altered later.

Similarly, software stability [3] is a measurement of how much effort is required to extend an existing application to meet new user requirements or slightly adapt an existing application to satisfy modified user comments (possibly after using the application for some time). Software stability includes reusability, extensibility, separation of development concerns and possibly software metrics.

Software development includes stages such as Domain analysis, Requirements elicitation and specification, Design, Implementation and Testing. However, only implementers have always been accused of wasting the company resources when developing non-reusable code that suffers from subjective view and/or un-explainable segments. All these problems make both the client and the service provider pray for a miracle to guarantee success in maintaining the developed software application.

The problem is that we usually rush into developing what we perceive from explicitly stated requirements, before completing the requirements themselves. What we typically see from the requirements is like the visible 10 percent of a floating iceberg, while a good 90 percent is still invisible, deep underwater. The overseen requirements analysis usually makes us deviate from what to develop (the whole iceberg) to how to develop the explicit stated requirements (the apparent 10 percent), and finally ends up searching for the keys to stability in the wrong place. It might be likened to a number of architects who contest and argue how to build a tower, while forgetting all about the foundations of the building. Thus, the right place to look for the lost stability keys is the requirements stage.

Coming back from philosophy to focus on the materialistic software world, Object Oriented Analysis and Design (OOA and OOD) have gained the trust of software developers. What lent a hand to OOA and OOD was the appearance of unified modelling language (UML) [1]. OOA has a key concept of assigning the responsibility of achieving the application functions into a number of interacting objects.

Although the object identification process has always being focused on explicitly stated requirements and information gathered from the stakeholders, the stage is ready for OOA when choosing the suitable objects.

In conclusion, OOA is a rich soil that can grow good crops when delivered with the right seeds. The ripe fruits of rightly planted seeds will be ready for gathering during the whole maintenance period. But this demands that the right seeds should be prepared and watered during the requirements analysis time. In particular, Object identification should attract more concentration by looking between the lines for requirements, and searching for the unstated supporting concepts behind the easily identified objects.

In a recent workshop the participants shared their research ideas about design patterns, stable architecture, and software metrics. My initial intuition was that the keys for software stability and reusability were hidden somewhere on requirements street [2].

What I realised was that the essence of software stability and reusability is related to the concept of knowledge normalisation. Probably that was a memory of a recent RESG seminar [6]. Stamper and Ades presented an interactive example of a school information system. The seminar organisers shared their ideas about MEASUR, semantic knowledge normalization and how this could contribute to the ease of extending developed applications.

To bring the idea down to earth, let's see what we can analyse from a bank system that has a single type of client account. The basic client account would have a single human owner. The problem is that sooner or later the bank may extend its business (to support different account types) or possibly another bank (with different account types) might contract the same software service provider. Thus, the single human client account would need to be extended into company-owned, multiple-owner accounts, etc.

Therefore, planning for a generic account would be a good idea, especially from the software firm's point of view. Such a generic account could be owned by a company or one person or more. Again it is a matter of what to develop rather than how. In addition, at the time of the contact, the bank will not ask for any general accounts or even mention anything about different types of account (remember this is part of the hidden 90 percent). Arriving at the generic account (what is termed in [5] the any-account pattern) is

feasible when considering carefully an instance of some specific account at the bank.

Splitting the observed account into account concept and variant account matching the current needs in one bank or another would be appreciated sooner or later when the bank needs to extend its business and needs to modify the sold application.

Although these ideas are not innovative, dating back to concepts of normalising semantic knowledge, software stability is attracting interest among both industry and research workforces. As proposed by Fayad, the idea has a great business value of developing applications in two stages: 1- expert stage to develop a product-line like analysis and 2- easier stage that can be practised by normal software engineers to tailor the product-line to the current client needs. This does not entail moving to a new approach like agile or aspect-oriented development. So, the time and cost of transition to a new technology would be saved.

In conclusion, the keys to stability are still hidden at the requirements analysis level. Some innovative analysis needs to be practised to reveal the whole requirements iceberg, before commencing the development lifecycle. Having the keys to stability at the requirements level would move us to a stage where the quality attributes of the developed software would be planned (and hopefully monitored) from the early stages. Possibly this will lead us to development process that guides the implementer to guarantee producing code of high quality, free from implementer personal views and/or own experience.

Bibliography

1) G. Booch, J. Rumbaugh, and I. Jacobson (1999), *Unified Modelling Language User Guide*. Addison-Wesley.

2) Islam AM El-Maddah (2005), *Planning Stable software applications using goal driven requirements Analysis*, Software Stability: Timeless Architectures and Systems of Patterns, The 3rd ACS/IEEE International Conference on Computer Science Systems and Applications, University of Nebraska-Lincoln, Computer Science & Engineering Department, Technical Report No. 04-12-03.

3) M. E. Fayad and A. Altman, (2001), *An Introduction to Software Stability*, Communications of the ACM, Vol. 44. No. 9, September 2001.

4) G. H. Galal-Edeen, (1999), *On the Architectonics of Requirements*, Viewpoint, Requirements Engineering 4(3): 165-168.

5) Hytham S. Hamza and Mohamed E. Fayad, (2004), *An architectural pattern for developing renting systems,* In Proceedings of 3rd Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP04).

6) Ronald Stamper and Yasser Ades, (2004), *Engineering Organisational Solutions from Human Information Requirements*, an interactive workshop at University College, London, UK.

A Very Fragile Coffin?

by Ian Alexander

One of my professors, Torkel Weis-Fogh, long ago described human-powered flight as "a study in the design of very fragile coffins". (For an example of Weis-Fogh's fascinating research work, see *Unusual Mechanisms for the Generation of Lift in Flying Animals*, Scientific American, November 1975, 81-87.)

The engineering of an aircraft with wings large enough to support the weight of a human body, but light enough not to make the propulsion of that airframe plus human power plant impossible for even the fittest athlete, is now a solved problem – just.

"The original Gossamer Albatross is best known for completing the first completely human powered flight across the English Channel on June 12, 1979." (Beautiful NASA photographs of the Gossamer Albatross and related craft can be seen at http://www.dfrc.nasa.gov/gallery/photo/Albatross/HTML)

Gossamer Albatross: A Very Fragile Coffin?

The Internet is a single gigantic system of systems. It was brought to its knees on the evening of 2nd November 1988 by a self-replicating program, a 'Worm'. It spread across thousands of computers in a few hours. The Worm's author apologised, but a precedent had been set. (Full details can be read at http://world.std.com/~franl/worm.html)

A power failure rippled across Italy in October 2003, blacking out 57 million people, including a million on the streets of Rome, who were gathered for an all-night cultural festival hubristically called White Night. At least 30,000 people were stranded on 110 power-deprived trains.

The event was caused, according to GRTN, by a tree's falling on a power line in Switzerland: Italy imports nearly 17% of its electricity from Switzerland and

France. Once one power source is disconnected, the mismatch of load and demand causes generators to slow from the nominal 50Hz. The resulting mismatch in phase between areas is serious, as out-of-phase waves cancel; so, automatic protection trips out each area in turn. (Full details from http://www.electricityforum.com/news/oct03/italyblackout.html)

It is said that only 30 to 50 people on this planet could now design a processor like the Pentium. Let's hope there isn't a coach crash on the annual chipmakers' jamboree.

According to Scientific American's June 2004 issue, a nuclear explosion in orbit would create such a shower of energetic electrons that communications, navigation, and observation satellites in low earth orbit would be disabled. Furthermore, none could be launched for up to five years, as the electrons would fall to earth only very slowly.

Are we not building "a very fragile coffin" for ourselves? Perhaps we are missing a few Robustness Requirements somewhere.

RE-verberations

This section is for items of news that have a bearing on requirements work. RQ would like to hear of such things from its readers.

Outsourcing: Turning of the Tide?

The Cutter Edge (1 February 2005) contained an article by Wendell Jones about what should and shouldn't be outsourced. Perhaps the mad rush to move everything offshore is coming to an end, as people start to realize that cutting wage bills may conflict with other business goals, such as providing a friendly personal service, or creating systems that meet their requirements.

> "A sourcing model that combines offshore, near-shore, and onsite delivery of software is emerging as the dominant model for global companies…. Some types of work, however, are best suited to stay onsite, while others can logically move offsite.
>
> Functions best for onsite work include: …
> - Requirements definition
> - Prototyping …
> - Usability [and]… Acceptance testing
> - User training
> - Implementation/cutover…
>
> Some of [these] functions… could be performed offsite, but are best done onshore or near shore. Closer proximity can facilitate spontaneous communications and relationship building…"

The Cutter Edge is a weekly e-mail service for IT professionals, provided free by the Cutter Consortium. You can register for your own free weekly subscription at http://www.cutter.com/research/email.html

–––––– oOo ––––––

The following piece is a description of the capabilities of a new RM tool from a leading vendor. RQ hopes that this style of article will be of interest to readers, and invites other tool vendors to submit similar descriptive articles on their offerings.

RQ's policy is to avoid advertising and similar materials, though enthusiasm for the solutions being offered is of course acceptable. Articles must be brief and must focus on the benefits that will be delivered to users.

Opinions expressed and claims made by guest contributors are not necessarily those of RQ.

Extending the Reach of Requirements Management

Jeremy Dick, Principal Analyst, Telelogic

Later this year, Telelogic will be releasing a new member of the DOORS/ERS family.

DOORS started life in 1992, and in version 7 today has become the world's most widely used requirements management tool. It owes its success to a unique ability to identify, organise and trace statements of requirement while retaining the vital concept of the requirements document.

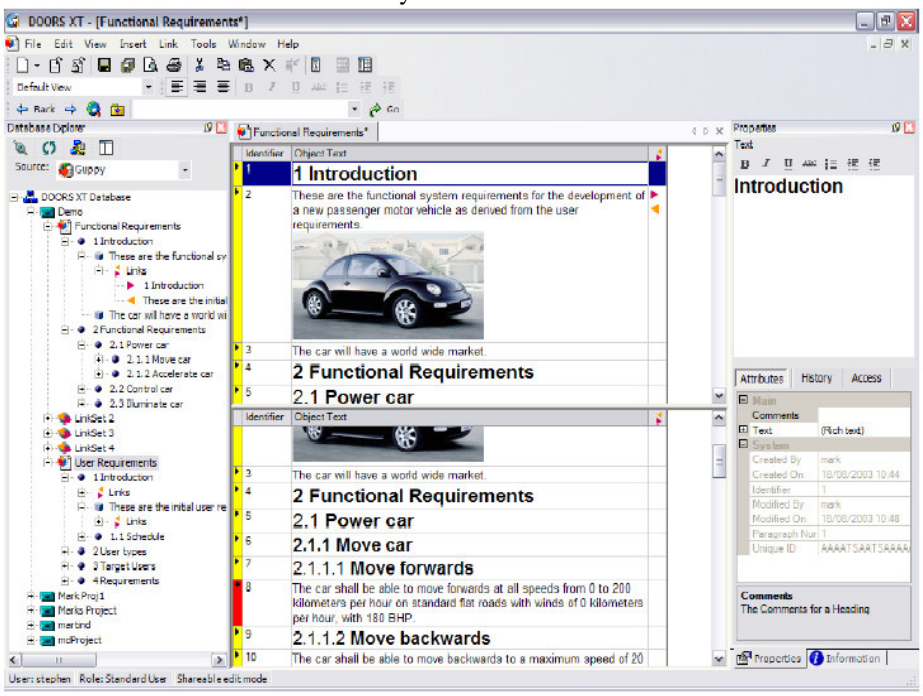
At the same time, the tool can be configured to support many flavours of review, change, agreement and validation processes. It is a highly flexible tool, providing a comprehensive API for customisation to meet more complex needs.

The new member of the family is DOORS XT. It is designed to address the requirements management needs of today's fast-paced, distributed, worldwide projects. More and more enterprises span several time zones, using distributed teams to achieve 24 hour working. Such environments stretch the capabilities of the current DOORS architecture beyond its limits.

To address these needs, DOORS XT offers a 3-tier architecture based on Oracle, WebLogic, and a browser-like thin client, easily installed and deployed via internet download. Remote access to a central database is possible over low bandwidth connections. Database backup and administration will be possible without interrupting 24x7 working. Customization will be possible through a set of APIs using industry standard programming and scripting languages, such Java, Java Script, C# and Visual Basic.

The DOORS XT client uses the latest .NET technology to provide an interface with all the look-and-feel and features of a modern Windows-base tool. And yet users familiar with DOORS 7 will immediately recognize and be comfortable with the DOORS XT interface.

The screen-shot below shows a number of the features.



On the left is the database explorer, which not only allows the user to traverse the hierarchy of folders and documents, but also to navigate within documents, right down to individual links on objects, all within the same explorer tool. In the middle panes are the open documents. In this example, two view are open on the same document. The change bar (with its characteristic traffic-light colours) and triangular link indicators are present in their own movable columns. Object properties are displayed in panes on the right.

A new concept of user-definable object types is implemented, with attribute definitions specific to each object type. All the classic DOORS history features are retained, with a complete audit trail of changes. Sharable access to documents is completely transparent, providing automatic locking at the object level.

From the outset, interoperability is provided between DOORS XT and DOORS 7. Data can be brought from DOORS 7 into DOORS XT, displayed, edited and returned.

DOORS XT has been created in Telelogic's Edinburgh development centre by a sizable team working over the last 2 years. Alpha and beta versions of the product have been on trial for the last few months, and the first full release is scheduled for the 2nd quarter of 2005.

Whether or not the world's best requirements tool has just got better, I will leave to you to judge …

RE-flections

Systems Engineering: -Ilities for Victory

I don't generally read history books, but at a recent meeting Stuart Arnold, instigator of the Systems Engineering Standard ISO 15288, and a leading light in the defence engineering community, showed me his copy of *The Most Dangerous Enemy, A History of the Battle of Britain*, by Stephen Bungay (Aurum, 2000). He said it was very interesting from a Systems Engineering point of view. (Since he showed me his, I showed him mine: I was reading Tom DeMarco and Tim Lister's *Waltzing with Bears*. There's a review of it on my website.) So, I read it with interest.

The first casualty of war is truth, it's said, and both sides had good reason to create myths for the Battle of Britain – whether of Aryan warrior-heroes with magic Messerschmitt Bf 109 swords, or of plucky little Britain, defenceless and unprepared, muddling through to fight off the 'Hun' through improvisation, daring, and happening to have ordered 1000 Hurricanes and 450 Spitfires, in peacetime, and trained their pilots, along with an Observer Corps of 30,000 men, and a continuous network of low- and high-level radar stations. Plainly the truth is very far from the myths, though as one pilot said, 'never have so many owed so much to so few' was true enough of the pilots' mess bills.

There was, in fact, a carefully-engineered air defence system in place in Britain in 1940, a rock on which airborne attacks could break like water. The Luftwaffe didn't know about it, and it's just as well, or they might have prioritised better.

The high-level radar network consisted of a chain of about fifteen tall thin steel pylons, sited from Norfolk to Devon, with a similar number of low-level radars. Each had a room nearby for the operators, who were trained to derive bearing, range, size of target, and (very roughly) altitude from the wiggly signals.

Radar detections and visual observations were telephoned back to sector command (one sector covered the Thames and east Kent, from Southend to Folkestone, for example). Sectors phoned group command: 11 Group, which through its location in the South-East had to do most of the fighting (under the inspired command of Keith Park), was at Uxbridge. Groups phoned Fighter HQ at Stanmore. Decisions were made at HQ and by Groups, and orders were given to the squadrons, which were stationed at many small airfields.

Now this, although entirely analogue, is a robust distributed system. The loss of any one radar or airfield, for instance, or any command post other than Fighter HQ (which was in a deep bunker), could be covered by its neighbours. The system could have been brought down by destroying all the radars and several control rooms, but this was hard to do. The radar towers were small and strong, and just swayed in the blast when bombed; the Luftwaffe didn't even suspect that sector and group control rooms existed, and in any case these had backups – one was in a butcher's shop.

If you're into -ilities, the system's survivability and controllability were excellent. Its performance was also remarkably fast, through a simple hierarchy of point-to-point links: telephone lines. The Luftwaffe did manage to bring down parts of the system once or twice, mainly by accident: a bomb cut off much of Kent's power grid, and several radars went down as a result; but not for long. The warning network worked splendidly; a wall of fighters met the Luftwaffe's every move. That alone had a powerful demoralising effect.

The other, and far better known aspect of the system is its teeth: the fighter squadrons, and the aircraft themselves. On 1st July 1940 there were 48 squadrons of Spitfires and Hurricanes, each of about a dozen serviceable aircraft, with others under maintenance, and about eighteen pilots. (There were also some Blenheims and Defiants.) Of those squadrons, by the way, eleven were up with 13 Group in Northumberland: squadrons were rotated to give weary pilots a rest, so there was always a reserve (whereas the Luftwaffe units fought continuously until exhausted).

The force was supported by many thousands of fitters and aircraftmen to keep the aircraft serviceable: between a quarter and a third of the planes with the squadrons were in maintenance at any one time. A steady stream of new pilots came out of flying training each month, though they had, horrifyingly, no training in fighting (and next to no gunnery practice, either); human operators are vital parts of most systems.

What is more, through the summer of 1940, the British aircraft industry was producing more than 400 fighters a month: twice what the Luftwaffe believed, and in fact easily enough to replace all the RAF's losses – Fighter Command had more operational planes, and more pilots, in September than in July. The system had depth.

The design of the fighters themselves was far from a happy accident or the work of one or two heroes, like Hawker's Sydney Camm who designed the Hurricane, Supermarine's Reginald Mitchell who designed the Spitfire, and Henry Royce (of Rolls-Royce) who led the development of the Merlin engine which powered both planes. The Air Ministry itself had issued specifications to industry throughout the 1930's, causing numerous prototypes to be developed.

Squadron Leader Ralph Sorley's calculations and gunnery experiments led to the specification of the almost unbelievably heavy armament of 8 guns (Brownings) for each fighter, to ensure sufficient lethality with what he rightly believed would typically be an opportunity to fire of no more than two seconds. Extensive debate on many such issues within the Air Ministry led to the planning and construction of the entire air defence network.

The fighters, with monoplane wings containing the guns, retractable wheels, and less obvious things like reliable VHF radios, were state-of-the art. The Hurricane was built very simply with a metal frame, built out with wood and a canvas covering, making it strong and easy to manufacture (about 5000 man-hours per airframe).

The Spitfire
A perfect balance of -ilities?

The Spitfire had the famous elliptical wing, giving greater speed and manoeuvrability (perhaps the most critical -ility of all). This was at a price: 13000 man-hours per airframe. Willy Messerschmitt had optimised for speed and manufacturability (some call it producibility): only 4000 man-hours; but the Bf 109

was no faster than the Spitfire, and was consistently out-turned by it. Heinkel had considered the elliptical wing, but rejected it as too difficult to manufacture.

In the event, Germany only managed to produce about 200 fighters a month in the summer of 1940: not enough to keep up with their losses. Heinkel and Messerschmitt were right about the need to minimise manufacturing effort: if they had made more labour-intensive machines, such as four-engined heavy bombers (as many armchair warriors have suggested), they would have run out of planes and aircrew even sooner. We, on the other hand, can thank our lucky stars, and Mitchell, that no-one tried to do Value Engineering on the Spitfire's wing.

There are plenty of other interesting design choices, but too many to mention here. It is often said that the cannon armament of the Bf 109 was superior to the Spitfire's, but the cannon's heavier calibre meant carrying only seven seconds' worth of shells, as opposed to sixty seconds' worth of bullets; and the vibration was severe and difficult to manage. Sorley's guns seem to have achieved higher lethality than the Bf 109's cannon in practice (number of planes shot down divided by number of bursts of gunfire). So that design trade-off was probably on the Spitfire's side.

The Spitfire went on to be manufactured all through the war in many versions, getting heavier and faster in the process (as fast as the early jets). In this sense its design was robust – in the face of unknown extensions and developments. It is difficult to write requirements for this kind of robustness, sometimes called modifiability or extensibility. As a result, these requirements are usually watered down or dropped altogether, but as the Spitfire showed, modifiability can be a war-winning -ility, along with sheer performance.

Dependability also demands mention. The Spitfire's build quality was far above the average for allied equipment at that time; as Bungay drily remarks, most allied equipment was much worse than the other side's. One might contrast the allied Sherman tank with the fast, powerful, and well-armoured Panther, for instance. The Sherman's grisly nickname was Ronson (a cigarette lighter): 'one strike and it lights'. Lest anyone should think this anti-American, let us give an all-British example: the Churchill tank. The driver could only exit when the gun was not pointing forwards, over his hatch: distinctly bad news when the tank was 'brewing up'.

The Spitfire was successful for many reasons, but one was that it always did what the pilot commanded with the stick, requiring only light finger pressure. In a tight turn or roll, the inner wing started to stall before the outer wing, causing a noticeable vibration, thus reliably warning the pilot. This in turn enabled every pilot to fly the plane to its limits – a life-saving feature. This wonderfully intuitive behaviour, 'washout', was created by a slight twist in the wing. Dependability, reliability, manoeuvrability, lethality, speed, controllability: no wonder the pilots were happy to have 'beer, women, and Spitfires'.

The Luftwaffe never recovered from its losses in the Battle. In September Goering went to the Pas-de-Calais to try to sort out the trouble, turning on all his undoubted charm.

Goering: What can I do for you?

Moelders: Upgraded engines for my Bf 109s.

Galland: A squadron of Spitfires.

It is said that Goering then lost his temper.

———————————————

This wasn't meant to be a book review (and it isn't one). Bungay has however organized an astonishing wealth of accurate information to tell the story properly with hindsight, for no-one at the time really knew all of what was happening, though Keith Park suspected most of it.

A cover quote unflatteringly says 'the most exhaustive and detailed account…'*, thus totally failing to make the point that a large pile of pieces doth not a system make.*

The book, however, for all its failings of style and repetitiveness, does brilliantly turn the evidence into a coherent story, blowing away a cloud of myths in the process.

Oh, and who was 'The Most Dangerous Enemy'? Britain, of course. Oberst Beppo Schmid, head of Luftwaffe intelligence in 1939, was right about that.

A Historical Proverb

After all that history, this issue's proverb is, naturally, historical. Ironically, it is frequently misquoted.

Those who cannot remember the past
are condemned to repeat it.

George Santayana

Life of Reason,
'Reason' *in* Common Sense, ch. 12, 1905-6

Or applying it to our case:

Those who have never heard of good system development practice are condemned to reinvent it.

RE-Creations

To contribute to RQ please send contributions to Ian Alexander (ian @ scenarioplus.org .uk).

Submissions must be in electronic form, preferably as plain ASCII text or rtf. Deadline for next issue: 15th June 2005

RE-Publications

Agile or Analytic?

User Stories Applied For Agile Software Development

by Mike Cohn, with a foreword by Kent Beck
Addison Wesley, 2004

Systems Modeling & Requirements Specification using ECSAM

by Jonah Z. Lavi and Joseph Kudish
An analysis method for embedded and computer-based systems
Dorset House 2005

Every now and again a pair of books comes on to the market offering interestingly contrasting perspectives on the same area. Since Kent Beck's dramatic Extreme Programming burst onstage in 2000, there has been a stream of books from the object-oriented software community on the theme of first 'extreme' and now 'agile' development approaches. These have argued, sometimes roughly and forcefully, that the traditional software development life-cycle was too cumbersome, and often ineffective. Meanwhile, the more traditional requirements engineering and systems communities have continued to do their own thing, tutting politely at the behaviour of the young bolsheviks outside torching their institutions.

Cohn's readable book offers a practical introduction to the use of brief Scenarios in the form of Kent Beck's User Stories. A user story is just what it says on the tin: it is a short, informal description of how some class of user could interact with and benefit from the proposed software. Isn't that exactly what Ivar Jacobson meant to do with his Use Cases? Sssh! Of course, both Scenarios and Use Cases now have (a range of) different meanings. Cohn is careful to distinguish User Stories from these in Chapter 12, 'What Stories Are Not'. Something that is particularly welcome is recognition that there are many kinds of user; Cohn suggests brainstorming possible kinds, perhaps an odd choice of technique here, but much better than just assuming that all users are rather like programmers.

And that is not all; Cohn shows in a practical way with examples and advice how to use stories to drive software development, so the book in fact covers all the early part of the life-cycle, and even strays into XP to show how the whole cycle works.

But instead of shouting in the streets, Cohn ventures inside the requirements literature, and reflects in a fresh and practical way on what is good and not so good in the tradition. First to go is vague talk of "elicitation" or "capture" (several well-known books are criticised at this point): requirements don't run about by themselves, nor are they waiting fully-formed inside people's heads just to be elicited. The Robertsons' "trawling" is however taken up with enthusiasm: you won't catch all the requirements in one trawl, so the process is inevitably iterative – which chimes well with the agile theme. We may argue that we knew it was iterative already, but far too many projects were not; and it is really welcome to see the Agile community engaging with the ongoing debate. However, Chapter 12 of Cohn also attacks traditional IEEE 830-style requirements as

> "tedious, error-prone, and very time-consuming"

and

> "quite frankly, boring to read"

so Cohn certainly hasn't become all soft and fluffy about requirements just yet.

Lavi and Kudish have been working on and with their method since 1980, initially in the aerospace industry but with a substantial backing in research. That both David Harel (inventor of the State Chart) and Mike Mannion (a pioneer of software engineering for product lines) liked the book enough to write cover blurbs was a futher clue that the contents would prove more appealing than the noun-heavy, acronym-laden title.

Lavi and Kudish cover a lot of ground. They begin quite traditionally with a context diagram, scope, and "determination of the system's boundaries". They make a data dictionary and a top-level specification. Then they examine modes and states, events, transitions, conditions and time – in an event-driven world like aerospace, this makes sense; indeed, one of the few books that deals thoroughly with events, Bill Wiley' s*Essential System Requirements*, comes from the same domain. But Lavi & Kudish go much further.

Their method is to step iteratively (Cohn and Beck have no monopoly on that front) through identifying the externally-observable ("E-level", black-box) system capabilities – such as one might record in user stories: they use operational scenarios, among other things; modelling the processes underlying those capabilities; and then going inside the box to model "S-level" or white-box system behaviour, and successively decomposing into subsystems that can be built (or purchased, or subcontracted) separately. To do that they describe methods of decomposition, including objects; they analyse the system's internal information flows and subsystem capabilities; they identify internal system modes and processes; and they describe the transition to design.

As if all that wasn't enough, they also examine the stakeholders' requirements process – like Cohn, they pay proper attention to the needs of different kinds of stakeholder, but quite unlike Cohn they know all about contractual and non-contractual situations, which they discuss in detail. They also consider what model-

driven requirements allocation and derivation (expansion) mean in practice. Like Cohn they know that requirements must drive design by insisting on verification, and though their language differs greatly, they know that a specification is only as good as its acceptance criteria. They end with a short chapter on requirements management, covering traceability and configuration management perhaps rather briefly. The value of the book is greatly enhanced by the presence of no less than five detailed case studies for exercises. An appendix summarizes the proposed notation; there is a detailed glossary.

One might sum up the difference between these two books by saying that where Cohn is surprisingly well-read and reflective for such a young approach, Lavi & Kudish are surprisingly modern and agile for something so traditional.

Both are excellent books – rich in detail, well thought out, closely argued, well illustrated, reflective and practical. That the approaches they advocate are so different says something about the state of software engineering today. Partly that reflects the different types of system that they have in mind: Cohn is in the main envisaging software that consults a database over the web, while Lavi & Kudish discuss military aircraft among many other (fully-worked) examples. Clearly the latter require decomposition of system requirements into subsystem specifications, so a hierarchical architecture is necessary, and documentation cannot be avoided. Perhaps Cohn does not address this issue head-on, given the domain in which he works: how far can agility be applied when a design and interfaces have to be agreed and shared among many subcontractors? Few things are less agile than contractual boundaries.

Both these books are immediately useful in industry, and can be warmly recommended. Students will also find them excellent companions – both are well-organized and indexed, both have good bibliographies, and both (surprisingly) provide exercises.

RQ doesn't often review HCI books, but since Søren Lauesen is the author of a wonderful RE textbook (see the review in RQ20 *(March 2000)), and has managed the prodigy of authoring a masterly HCI text as well, readers might perhaps find the following book interesting. There is, of course, a significant overlap between the two disciplines.*

User Interface Design Styles and Techniques

By Søren Lauesen
Addison-Wesley, 2004
(ISBN 0321181433)

> There are thousands of books on programming. Common to all those I have seen is that the user interface is rather unimportant - it is just a matter of input to and output from the program....
>
> There are a few dozen books on HCI. They tell a lot about human psychology... [but] say very little about the actual design of real-life user interfaces.

It's interesting to see a detailed practical cookbook from the HCI domain, as there is plainly a significant overlap with requirements work. This textbook is clearly also a huge amount of work from one author, based on a lifetime of experience; and it is all the more remarkable coming from someone who has also written a requirements textbook! Lauesen is as always clear, practical and helpful, taking care to explain things even when they might seem obvious (that's where the reader's assumptions are).

The book is a detailed step-by-step guide to designing user interfaces. It looks (to someone from outside that domain, except in a very small way) like a valuable text for students, and a helpful reference for novice HCI designers.

Each chapter begins with a summary and 'highlights'; the chapter body supports the text with plenty of graphics, tables, and screenshots; at the end there is a 'test yourself' list of a dozen questions (when appropriate). There are design exercises at the back, covering all the chapters, and four 'projects' for further study. There's even a formidable '4-hour written exam' paper. But answers are not provided.

Part A covers classic Usability, Prototyping & Iterative Design, Data Presentation, and Mental Models, with illustrated theory on what works for the human brain and what doesn' t.

Part B, Systematic Interface Design, covers Analysis, Virtual Window(s) Design (the UI concept, not the operating system), Function Design, Prototyping, and reflection on User Interface Design.

Part C, unexcitingly titled Supplementary design issues, looks at more advanced stuff, giving detailed examples, and covering issues such as user support, usability testing, heuristic evaluation (ie 'someone looks at the user interface and identifies the problems'), systems development and data modelling.

The User Interface at its most Delphic

Lauesen offers a wonderful analysis of 'an error message from Microsoft Word' (97 in fact), using the book's Heuristic Rules to good effect. When you have put in an invalid line measurement (eg if you type in '0.5 lines' rather than '0.5li'), which you aren't prevented from doing, you get the cryptic warning "This is not a valid measurement", complete with bright yellow warning triangle containing a big black exclamation mark – an oddity, as the rest of Word 97 is pretty friendly. The warning fails on at least 5 of 9

heuristics, and its scores on some of the others are a bit doubtful too.

> How can it happen?
>
> This example from Word is quite scaring, but also very common. In Office 2000, the error message is still the same, except that there is no Help button in the message anymore...
>
> How can it happen? The reason is that error messages and help texts are made by technical writers - often at the last moment before release ... but they [don't] understand what the system does.

Are you sure your system will work better than that? If not, let Lauesen be your doctor.

The book is well-organised and indexed, with a specially good bibliography that summarises the message of each book or paper. It feels quite different from any HCI book I've seen. Perhaps that's because it's so much more practical.

RE-Sponses

RQ welcomes comments and reactions to articles and reports published in its pages.

Decomposition or Elaboration?

Ian Alexander's piece in issue 34 [1] questioned the assumption in classical RE theory, which he neatly summed up as URD <-> SRD, that a SR (System Requirement statement) is valid if and only if it is called out by a UR (User Requirement statement). It is a nice tidy concept. Such a requirements model is amenable to proof of traceability, and might even help to generate lawyer-proof documents, but does it square with reality, and how much value does it add?

By coincidence, a few pages later, in the review of *Requirements-Led Project Management*, chapter 5, he says "...the provocative title 'Inventing Requirements'. Of course, it's something you' re not supposed to do ... but the authors argue convincingly that analysts should invent ...".

This juxtaposition resonated with me, because I have long held that 'requirements decomposition' was only half of the process (perhaps the more obvious part) and that 'requirements elaboration' was a much more helpful term. Both are about adding detail, but whereas one assumes that the detail is somehow encapsulated in the input statements, simply awaiting analytical unfolding, the other faces the reality that even if the input words are nominally complete (which they are likely not to be) they do not encapsulate the full requirement without also understanding how they relate to the often unstated contextual detail of the user organisation and how it works. Nor can they hope to encapsulate answers to the many technology related questions that must be answered in order to formulate a workable SRD (System Requirement Document). Does this mean that traceability back to input requirements is an illusion? Surely not —traceability is 'a good thing'.

I believe the answer lies in the distinction between the process of requirements engineering, and the desirable end state to which it is directed. RE is a collaborative effort between a number of players. At its simplest there is a problem owner, a 'solution owner', and an analytical go-between —the requirements engineer. In practice, the players can be groups of slightly disparate stakeholders. For example the problem owners could include end users, process owners and funding providers. The 'solution owner' is responsible for transforming the requirement into the solution space, whether or not he provides the solution.

The objective of the process is to produce a tractable and manageable set of requirements for a plausible solution that can solve (an agreed subset of) the problems. For a large system you won't be confident that you have done that without a fair degree of explicit traceability between a mature set of statements representing the problem (URD) and a set of requirement statements representing a credible solution (SRD). The question is how you get there.

It seems unreasonable to expect the first stab at the URD to be either complete or consistent, and certainly not to contain all the hooks (other than very high level statements) for traceability of all that ought to be in the SRD. Anyway, the URD ought to be simpler than the SRD. The SRD can't be produced without an understanding of the constraints that the problem space imposes on the solution space, nor I contend can the (final) URD be produced without an understanding of the constraints that the solution space imposes on the problem space. Thus to a degree, the RE process of building a bridge between URD and SRD is a two way one, and we should expect some URs to be challenged, some hidden ones to be discovered.

At the end of the process, there should be a well understood, fully traceable mapping between URD and SRD, but it will not be one to one mapping, and nor should it be a 'content free' mapping. Years ago, I saw large parts of an SRD generated by mechanistically transforming the statements. (I forget the exact words, but the value added amounted to changing 'The user requires...' into 'The system shall provide ...'.) For some functional requirements, that might work, though one would expect differing levels of detail, but for the quality attributes (ie 'nonfunctional requirements') it adds no value.

One way in which SR thinking can influence the URD is via reasoning of the form 'Technology can't deliver X, so when trying to deliver user requirement Y, it will impose undesirable quality Z'. To illustrate this, consider the rather low level and somewhat dated

example of a requirement for spell checking. X might be the speed at which it is possible to create alternative spellings, Y might be the requirement for 'an interactive spelling checker', and Z might be the usability penalty of several seconds disruptive delay between each user decision on spelling (yes, some of us can remember it). The URD quite likely would not contain a requirement to avoid this particular usability nasty, but an effective requirements engineer would extract enough insights from the other players about how the spell checker might be used in the organisation, to detect that Z is implied by overall expectations about quality in use, and also that meeting it might be an issue. The problem owner can then consider whether to adopt Z in the URD, or to revisit Y to see whether the business need could perhaps be met by an altered requirement Y*.

Another way that the URD might be influenced is of the form 'Technology could help meet constraint A by providing function B (which not currently required)'. To illustrate this, consider the requirement for a command system where A is the need to contain operator workload within bounds and it transpires that a significant part of the operators workload is a supplementary task outside the defined system boundary, but which could easily be provided within it, for example the conversion of simple codes B. B is not required, and could be branded 'function creep', but it might make a more cost effective contribution to delivering A more than the usability engineering needed for the more complex core functions to be provided.

A collaborative, iterative model chimes will with much 'anti-waterfall' and 'anti over-the-wall' thinking, but can it fit within the widely used competitive model of procurement selection, where the customer provides the URD and the competing contractors offer their SRD (among other study outputs)? Traditionally, this contracting model has discouraged fruitful interaction with the URD other than 'clarifications', but it need not do so. Some recent MoD projects required contractor's observations on the URD as an early deliverable. Where contractors are competing not just on solution, but on competence and integrity for down-stream partnership with the procurer, there is every incentive to offer added value by suggesting improvements or extensions to the procurer's URD in this way.

So to answer Ian Alexander's question, whether an SR is valid if and only if it is called out by a UR – yes and no. There should be traceability as the end state, but not necessarily from the initial URD, and not one to one with each SR explicitly 'called out'. RE should add value by elaboration, not just by decomposition.

References

1 - What is an exhaustively satisfied user requirement anyway?, Ian Alexander, *Requirenautics Quarterly* issue 34, pp 9-10, December 2004.

2 - Review by Ian Alexander of *Requirements-Led Project Management*, Suzanne & James Robertson, in *Requirenautics Quarterly*, issue 34, p17, December 2004.

Correction

The ISBN of *Requirements-Led Project Management* by Suzanne & James Robertson, Addison-Wesley, 2004 (reviewed in RQ34) is 0-321-180623.

RE-Sources

Books, Papers

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

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

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

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

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

Requirements Engineering, Student Newsletter:
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/

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

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

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

Mailing lists

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

The RE-online mailing list acts as a forum for requirements engineering researchers and practitioners. To subscribe to RE-online mailing list, e-mail 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: A free mailing service for the table of contents of the *International Journal on Software Tools for Technology Transfer.*
http://link.springer.de/alert

RE-Actors: the committee of the RESG

Patron:

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

Chair:

Dr Pete Sawyer, Lancaster University, Computing Department, sawyer @ comp.lancs.ac.uk.

Vice-Chair:

Dr Kathy Maitland, University of Central England, Kathleen.Maitland @ uce.ac.uk.

Treasurer:

Prof. Neil Maiden, Centre for HCI Design, City University, N.A.M.Maiden @ city.ac.uk.

Secretary:

David Bush, National Air Traffic Services, David.Bush @ nats.co.uk.

Membership secretary:

Dr Juan Ramil, Computing Department, The Open University, J.F.Ramil @ open.ac.uk.

Newsletter editor:

Ian Alexander, Scenario Plus Ltd., ian @ scenarioplus.org .uk

Publicity officer:

William Heaven, Department of Computing, Imperial College, su2 @ doc.ic.ac.uk

Regional officer:

Steve Armstrong, Computing Department, The Open University, S.Armstrong @ open.ac.uk.

Student Liaison Officer:

Carina Alves, University College London, Department of Computer Science, c.alves @ cs.ucl.ac.uk

Immediate Past Chair:

Prof. Bashar Nuseibeh Computing Department, The Open University, B.Nuseibeh @ open.ac.uk

Industrial liaison:

Prof. Wolfgang Emmerich, University College London, W.Emmerich @ cs.ucl.ac.uk.

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

Gordon Woods, Independent Consultant, Gordon @ cigitech.demon.co.uk

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

RQ35 (March 2005)

The Requirements Engineering Specialist Group of The British Computer Society

www.resg.org.uk

Individual Membership Form for 2005

1. Membership Type

Please indicate type of membership:

BCS/IEE members, please

BCS / IEE member (£10) [ ] *indicate membership number*_____

Non-BCS / IEE member (£20) [ ]

Full-time student (free) [ ] *Studying at:*_____

If you need a receipt, please tick here [ ]

Corporate Membership also available – details at www.resg.org.uk or ask the Membership Secretary

Payment by cheque only. Please make it payable to "The BCS Requirements Engineering Specialist Group"

---

2. Your Details Title: Mr/Mrs/Ms/Dr/Professor/Other:_____(delete as appropriate)

First Name:_____Surname:_____

Address for correspondence:_____

_____Postcode:_____

Phone:_____Fax:_____

E-mail address: *Please write your e-mail address clearly using BLOCK CAPITALS*

As an RESG member your e-mail address will be added to the RESG mailing list

Optionally, indicate:

Your organisation's name: _____

Its type of business/domain:_____

3. Your Specific Interests

Areas of interest for the RESG given at *http://www.resg.org.uk/about_us.html*. Is there another area would you like us to add?

---

4. Your Participation Preferences

Please indicate what timings/duration for RESG events would suit you:

Whole day [ ] Half day [ ] Evening [ ] Other (please specify) [ ]_____

Please indicate whether you would be willing to help the RESG with:

Publicity [ ] Newsletter contributions [ ] Organisation of meetings [ ]

5. Your Mail Preferences

The RQ quarterly newsletter is regularly sent to all RESG members. It will be sent as a PDF e-mail attachment unless you prefer a hard-copy of RQ delivered by post. For hardcopy delivery, please tick here [ ]

I understand that the information supplied on this form will be held in a database and used for the purpose of distributing information relevant to the activities of the RESG.

6. Your signature:_____Date:_____

Please send this form with payment (if applicable) to: Juan F. Ramil
RESG Membership Secretary membership-RESG@open.ac.uk Fax +44 (0)1908-652140
Computing Dept., The Open University, Walton Hall, Milton Keynes, MK7 6AA, U.K.