Requirements Quarterly

The Newsletter of the

Requirements Engineering Specialist Group

of the British Computer Society

©2005 BCS RESG http://www.resg.org.uk RQ36 (June 2005)

Contents

RE-Soundings

From the Editor

In this issue, we report on the first-ever RESG event on i* and its very special take on goal and NFR modelling. Neil Maiden achieved the extraordinary coup of getting both Eric Yu, the creator of i*, to give a tutorial, and a programme of excellent speakers to relate their experiences of the technique. Let's hope there'll be many more such events in the coming years.

In the Papers section, we have interesting, practical and well-argued (though very different) points of view on how requirements should be done from Björn Regnell and David Gelperin.

Alert readers may have noticed a small change to the RQ masthead – we've simplified the name of the newsletter, and taken the opportunity to simplify the layout of the masthead as well.

Now is the time to get booking your trip to Paris for RE'05, including selecting the workshops and tutorials you are going to attend.

Ian Alexander,
Scenario Plus

Chairman´s Message

As I write this, REFSQ'05 (international workshop on Requirements Engineering: Foundation for Software Quality) has just finished. The usual strengths of REFSQ were on show; a healthy mix of people ranging from grizzled veterans like Dan Berry, Sjaak Brinkkemper and our very own Neil Maiden, to students in the early stages of their doctoral research. A significant number of the 24 participants were either currently or recently employed in industry or had a foot in both academia and industry.

One of the reasons I like REFSQ is because, along with papers reporting incremental advances, it's where you get to see new ideas on their first outing. This year, for example, we had papers positing how service-based systems and dynamically adaptive systems change the role of RE. Expect to see much more on this in the near future – in fact there's a dedicated workshop on RE for service-oriented systems at RE'05 (SOCCER – see *RE-Calls*).

There were also excellent papers on applying i*, on requirements and testing, and on RE for market-driven development. You needn't feel too disadvantaged if you missed REFSQ, though, since these are all topics that get an airing in this RQ.

Even if you missed REFSQ, make sure you don't miss the next big event in the RE calendar; the Tool Vendor's Day (see *RE-Treats*) on the 20th July. This also coincides the RESG AGM where you can let us know you feelings on our latest innovation: *Requirenautics* Quarterly has become *Requirements* Quarterly. We'll be going tabloid next …

Pete Sawyer
Computing Department, Lancaster University

RE-Treats

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

Forthcoming events organised by the RESG:

Tool Vendors Day and AGM

Wednesday 20th July 2005

The AGM this year brings an exciting free event: a Tool Vendors challenge. The RESG is delighted that several major tool vendors have agreed to rise to the occasion and submit their approaches to comparison and scrutiny.

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: 13th IEEE International Requirements Engineering Conference

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

http://www.re05.org

RE'05 is an exceptional opportunity to meet in Paris and share experience with worldwide requirements engineering academic and industrial experts. The conference presents a highly selective program of carefully reviewed papers. Besides, a unique forum of exchange is proposed under the form of 12 workshops and 9 tutorials. This is the occasion to learn about ground breaking requirements engineering methods, techniques and tools.

Programme of Workshops at RE'05

An exceptionally rich and varied programme of workshops is being run this year at RE'05:

WS1 : (REET) Requirements Engineering Education and Training. Tuesday August 30th. Organized by : Didar Zowghi (University of Technology, Sydney), Jane Cleland-Huang (De Paul University).

Effective Requirements Engineering (RE) is increasingly recognized as a critical component in the success of a software development project. This has led to a growing identification of the importance of incorporating significant RE components into the curriculum of university degrees in Software Engineering, Computer Science, Information Technology and other related areas. Furthermore many industrial organizations are recognizing the need to develop RE related training programs as part of their ongoing process improvement initiatives.

In addition to topics related to curriculum development, creative contributions related to pedagogical techniques for teaching RE skills are strongly encouraged. These skills include requirements elicitation, modeling, analysis, conflict negotiation, consensus building, and requirements specification writing, interviewing, and reviewing skills.

WS2 : (SREP) Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific RE Processes. Tuesday August 30th. Organized By : Jolita Ralyte, University Of Geneva, Switzerland, (Jolita.Ralyte@Cui.Unige.Ch), Naoufel Kraiem, University Of Manouba, Tunisia, (Naoufel.Kraiem@Ensi.Rnu.Tn), Pär J Ågerfa;l, University of Limerick, Ireland, (par.agerfalk@ul.ie)

WS3 : (REProMan) Interplay of Requirements Engineering and Project Management in Software Projects. Tuesday August 30th. Organized by : Andrea Herrmann, Sari Kujala, Marjo Kauppinen (Helsinki University of Technology), Soren Lauesen (IT University of Copenhagen), Barbara Paech (University of Heidelberg), James Robertson (The Atlantic Systems Guild).

WS4 : (REBNITA) Requirements Engineering for Business Need and It Alignment. Monday August 29th. Organized by : Karl Cox (National ICT Australia Ltd), E. Dubois (Public Research Centre Henri Tudor), Y. Pigneur (HEC Lausanne), June Verner, Steven Bleistein (National ICT Australia Ltd.), Alan M. Davis (University of Colorado), Roel Wieringa (U. Twente).

It is no longer possible to consider IT separate from the business organization it supports, and hence requirements engineering should address the business needs of an organization. Business needs can be described through IT alignment with business strategy, explicit value analysis of IT, integrated market analysis and product development, as well other types of analysis of business processes, organization infrastructures, business goals and objectives. Though it is recognised that requirements engineering (RE) is a natural bridge that connects the business world and the IT world, much of RE research continues to be solution-oriented and avoids addressing the hard, real-world business problems that confront business practitioners every day. This trend, if continued unchecked, threatens to ultimately make requirements engineering research of little relevance or importance to industry. As such, the goal of this workshop is to provide a specific forum for research that is motivated by requirements engineering approaches that encompass organizational business needs. Deadline for submission : 3rd June 2005.

WS5 : (DSD) Distributed Software Development. Tuesday August 29th. Organized by : Daniela Damian (University of Victoria), Schahram Dustdar (Vienna University of Technology).

WS6 : (SREIS) Symposium on Requirements Engineering for Information Security. Tuesday August 29th. Organized by : John Mylopoulos (U. Toronto), Gene Spafford (Purdue), Annie Anton (NCSU), Fabio Massacci (UTrento), Haris Mouratidis (UEL-UK) and Loris Penserini (ITC-IRST).

Engineering situations vary considerably from one software or information system development project to another. A number of different characteristics such as project objective, application domain, different features of the product to be developed, involved stakeholders, and various technological conditions and constraints have a significant impact on the Requirements Engineering (RE) process. As a consequence, each software or information systems project requires a specific RE method and tool to support the RE process. The question then, is how to select, adapt or construct, and manage (perhaps changing) requirements on a RE method that best suits the situation of the project at hand? Deadline for submission : June 17, 2005

WS7 : (SOCCER) Service-Oriented Computing: Consequences for Engineering Requirements. Tuesday August 30th. Organized by : Luciano Baresi (Politecnico di Milano), Neil Maiden (City University), Xavier Franch (Universitat Politècnica de Catalunya).

The service-oriented approach is becoming more and more popular to integrate highly heterogeneous systems. Web services are the natural evolution of conventional middleware technologies to support Web-based and enterprise-level integration, but the paradigm can also serve as basis for other classes of systems. To realize a service-oriented architecture we need techniques to identify and specify requirements on services in a machine-interpretable way to enable the dynamic composition and deployment of systems that meet the expectations of the different stakeholders. We need new capabilities to monitor the behavior of deployed systems and reasoning on partial matches, deviations, and corrective actions. We need to integrate service-oriented architectures with existing component- and COTS-based architectures that deliver capabilities not suited to services. And finally, we need to be able to exploit the availability of services to discover new opportunities that improve existing requirements processes and techniques. The workshop intends to provide an opportunity for the communities that work on requirements and service-oriented applications to meet and share their knowledge to set appropriate theoretical foundations, define special-purpose methodologies for requirements specification, and develop supporting technology.

WS8 : (CERE) Comparative Evaluation in Requirements Engineering. Tuesday August 29th.

Organized by : Alistair Sutcliffe (University of Manchester), Ann Hickey (University of Colorado at Colorado Springs), Vincenzo Gervasi (University of Pisa).

WS9 : (REDECS) Requirements Engineering Decision Support. Tuesday August 29th. Organized by : An Ngo-The (University of Calgary), Günther Ruhe (University of Calgary)

The growing importance of Requirement Engineering has also intensified the need for decision support. This requires a closely collaboration with researchers from other disciplines, particularly management sciences and decision theory. The workshop is an initiative to gather researchers and practitioners in an effort to promote the awareness of the role of decision support in requirements engineering. Its expected outcome includes identification of problems and challenges, directions for future research and, most importantly, collaborative initiations with researchers in management sciences and decision theory. Deadline for submission : 9 June 2005

WS10 : (RHAS) Requirements Engineering for High-Availability Systems. Tuesday August 30th. Organized by : Donald Firesmith (Software Engineering Institute)

WS11 : (FRU) How to model Firms' Requirements and make them Understandable by Business Directors, Information Systems & Process owners and IT managers. Tuesday August 30th (morning). Organized by : Jean-Marie Faureis (Crédit Agricole), Laurent Colletis (Mutuelle Sociale Agricole), Philippe Desfrayis (Softeam), Christophe Longépé (Société Générale), Jacques Printzis (CNAM), Lionel Ploquin (Ministère des Finances), Michel Volle (Club des maîtres d'ouvrage des systèmes d'information).

WS12 : (MEAP) Managing Enterprise Architecture Projects. Tuesday August 30th (afternoon). Organized by : Jean-Christophe Bonne (Renault), René Mandel (Oresys consulting).

Training Courses in October

Requirement Specification, 3-4 October; Evolutionary Project Management, 5-6 October; and Agile Inspection, 7 October. 10% discount for BCS & RESG members.

http://www.testing-solutions.com/gilb

DiSD 2005

The International Workshop on Distributed Software Development, August 29th, 2005, Paris, France

http://www.infosys.tuwien.ac.at/Staff/sd/DiSD2005/DiSD-2005.html

IWPSE 2005

International Workshop on PRINCIPLES of SOFTWARE EVOLUTION, in cooperation with ACM/SIGSOFT

September 5-6, 2005, Lisbon, Portugal

http://www.rcost.unisannio.it/iwpse2005

International Conference on Software, Telecommunications and Computer Networks

September 15-17th, 2005, Split, Croatia

http://www.fesb.hr/SoftCOM

ICSSEA 2005

18th International Conference on Software & Systems Engineering and their Applications

November 29, 30 & December 1st, 2005, Paris, France

http://www.cnam.fr/CMSL

RE-Readings

Reviews of recent Requirements Engineering events.

The i* Events

20-22 April 2005, City University, London

Modelling Your System Goals: The i* Approach

Wednesday 20th April 2005, City University, London

The meeting was fully-booked with an extra-large room for 60 delegates. We were all excited at this first i* event at the RESG.

Professor **Eric Yu**, University of Toronto, Canada gave the morning tutorial, entitled Strategic Actors Modeling for Requirement Engineering - the i* Framework. He is the father of the i* approach, and has been researching goal modelling for over a decade now.

Early RE

Yu began by asking what Early RE (Requirements Engineering) is. Traditional RE models (eg dataflow) what is happening, but not why; nor why we want it to happen some other way in future. Reasons may be stated but they are in words, and so are hard to analyse.

But different people have different desires and purposes. So, to understand a domain, you need to know about the range of different stakeholders, and their intentions. Reasons matter -- or else we'll solve the wrong problem, build unused shelfware, ignore changing needs and regulations, and fail to cope with internationalisation. Relationships among stakeholders matter; people have varying concerns like security, privacy, trust, profit, market positioning, IPR, and political advantage -- and these are all beside the normally-modelled functionality! We're failing to model explicitly how people can achieve what they want, ie to make RE goal-oriented.

The Harvard Business Review last year published a paper by Nicholas Carr arguing that "IT Doesn't matter", quoting some famous surveys. For instance, most current IT (Information Technology) projects fail to meet objectives (85% - Gartner), are delayed (average 18-24 months - Standish), cancelled (35% - Gartner), or develop functions that are never used (93% - Gartner). IT still matters, said Yu, but we need more attention to the front (early) end. Analysts deserve a pay-rise [laughter].

Yu argued that we need systematic methods, bringing modelling and reasoning support tools and traceability into RE. We need to explore what is possible, desirable, and viable.

What Models?

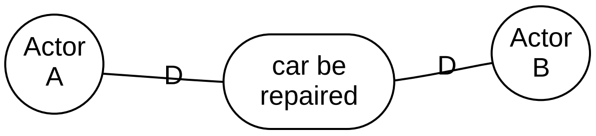
The UML doesn't support such things – it was born 15 years ago with Jacobson, Booch and so on, before the Web, in a simpler world with less socio-technical complexity. UML deals with techniques mainly for specifying and designing software. Perhaps it will gradually move further towards people and their requirements.

Successful modelling techniques are very simple: eg Entity-Relationships (2 concepts -- entities, relationships); or Dataflows (3 concepts -- processes, stores, dataflows).

So, what concepts should i* try to capture? He listed intentionality, autonomy, sociality, identity and boundaries, strategic reflectivity, rational self interest (quite a complex set). So the key concepts in the approach are Actors (human or machine), and relationships (strategic intent).

But i* does not look at the execution level (it doesn't run! something like Tropos can do that), nor time (it's not about sequences like scenarios). But stakeholders and goals underly all systems. It is also confined to what people want; it doesn't cover what people believe or assume, so it only deals with part of what is conventionally dealt with by domain rationale or argumentation models (or simply by requirement attributes that record justification).

Strategic Dependency (SD) is the basic i* relationship. Actor A depends on B for something (like getting his car repaired). A wants it; B can do it. A "D" is drawn on the line to show the direction of Dependency, and the line is labelled with a variously shaped box to show the nature of the Dependency. It could be for a goal to be achieved (rounded box), task to be performed (hexagon), resource to be furnished (rectangle), or softgoal (blob or cloud). (A goal doesn't care how the result is achieved; a task stipulates procedurally what to do.)



David Bush said this was fine if there was a perfect match between want and service (the garage can repair my car exactly as I want). But what if there was an imperfect match of bid and offer? Yu replied that he assumed that there was a commitment to achieve a match, so the two would converge. But unfulfilled goals inside an actor could exist (in a Strategic Rationale model), not met by a Strategic Dependency.

The audience also discussed the allocation of intentions to goal vs task; one could turn into the other; and of 'hard' and soft goals: wanting a 'fair settlement' of an insurance claim could be both a hard goal -- wanting compensation -- and a softgoal -- wanting justice.

Strategic Rationale (SR) looks at both the internal intentions of an actor, and alternative Dependency relationships with other actors. The internal intentions are shown by drawing a large circle for each actor, and then showing internal goals (etc) within that circle. Those actors appeared on the SD diagram with dependencies between them; the same dependencies are shown on the SR diagram, but now they relate intentions held by those actors. In other words, the SR model decomposes the broad relationships of the SD model into detailed intentions. Those could be said to be part of the psychology of those people (and this was criticised in questions from the floor); but Yu explained that we don't know yet what the system boundary is. Goals and tasks could be moved (delegated) to new agents once we'd decided to create them. That enables analysts to consider explicitly how to redesign the work, by varying the scope of the software (or other machine) under design. But the goals would remain stable so it makes sense to model them.

Richard Veryard (*see his article in this issue of RQ*) said that SR could vary between people in one situation (people wanting cars repaired, patients in hospital, whatever). Yu said we could make generic or even personalised models.

You can go much further; tasks can contribute positively or negatively to softgoals (shown as arrows labelled with + or -); softgoals can be related with + and -, or if that's not enough, graded into three levels of helping and three of hurting, with 'unknown' in the middle. Similarly you could show correlations for side-effects, also at seven levels. And, tasks can be decomposed into an AND/OR tree of subtasks (AND has a bar across each linking line; OR just has a plain line between the tasks). But all that might be a bridge too far: that is a lot of constructs.

Wasn't SR going into design decisions? Yu said that model elements had to come from stakeholders. Initially these might just be "be quick", "be low effort". But eventually these have to be operationalised into things that would actually be done.

Each diagram shows one model, which could be as-is, or one competing version of the as-should-be. i* could then represent reasoning about different alternatives. To prevent diagrams from getting too cluttered, it's best not to put alternatives on the same diagram.

On SR diagrams you can distinguish human actors from abstract agents, roles, and even (job) positions – a job may consist of several roles, and roles can be assigned to human agents or to machine/software agents. But this subtlety might be too much for many stakeholders.

Vulnerabilities can be identified by looking for loops in an SD model; if I depend on them and they depend on me, then we have to work together or the system will fail.

Conflicts can be identified by looking for competing demands on an agent from different sources.

An SR model can simply break tasks down directly into ANDed subtasks (as in conventional goal modelling). But it can also show ANDed subgoals, which can then be met by subtasks; this encourages people to think creatively of alternative ways of meeting those goals.

In conclusion, actors are semi-autonomous agents, only partially knowable, with wants, abilities, and intentional dependencies; they have the power of choice; and we can explicitly reason about (partially) alternative means to the ends that they desire. Relationships cannot be fully quantified or reduced to theories; relationship networks are unbounded; agent boundaries (and hence, their very identities) can change; co-operation cannot be taken for granted; and conflicts may be hard to resolve. Agents may reflect upon their own actions (and so, the development world is strategic -- at a meta-level -- with respect to the world of operations). All of this is a mile away from the assumptions of software developers, even when they use the language of agency.

–oOo–

In the afternoon we heard about some experiences of applying i* in real life.

Professor **Manuel Kolp**, University of Louvain, Belgium, spoke about a range of experiences with i*, including Hospital Bed Management and Steel Making as contrasting examples.

The Tropos process uses i* for early and then for late requirements. Even at design level, social concepts are used alongside technical (UML) models.

He described the analysis of the organizational needs of bed planning and management at the Saint-Luc Clinics of the University of Louvain (UCL). Requirements analysis was needed to reorganise and improve the hospital information system.

Complicating factors included the complexity and importance of the university hospital, the social individualism of its employees and units (e.g. doctors, nurses, staff, patients, medical units, administrative services), the types of medicine practiced, and changes needed to the environment, in particular to emergency and pathology hazards activities.

Bed reservation has to be co-ordinated with each health unit, which has to provide current bed occupancy every day. This can't be predicted as both routine and emergency (= unplanned) cases occur. Management turned out not to be involved in bed booking, as they never had timely information on patients and beds. Kolp decomposed each actor into tasks: those of the health unit, the bed booking system and so on. (He wisely varied the i* notation, using the label "hurts" to clarify that relationship between tasks and softgoals, eg that quality is hurt by manual copying of paper records.)

Another project was to reorganise UCL's information systems. This was modelled with "i*-compliant" KAOS method. He said that the goal decomposition in the two was the same. i* goes from SD to SR; then KAOS takes over for goal and obstacle analysis.

He also described the DesCARTES software design tool being developed at UCL that has been used to support the use of i* in different projects. It supports a mix of software design tasks including structural and behavioural modelling and code generation.

Dr **Anna Perini**, ITC-irst, University of Trento, Italy spoke on Understanding the Requirements of a Decision Support System for Integrated Production in Agriculture.

She said that apples and wine were very important in Trentino.

Decision-making in agricultural production, and more generally in the management of environmental problems, rests on different types of knowledge and data (e.g. knowledge of biological processes, geographical and weather data, knowledge on the use of chemicals and on the application of agronomic practices, local and national rules on product distribution) that are typically produced and made available by different organizations.

For example, the apple codling moth is a critical pest for apples, as it is resistant to pesticides. Qualitative models of codling infestation tend to fail in Trentino as the mountain slopes vary in height and orientation ('heterogeneous orography'); temperature is crucial for egglaying. So, it is necessary to know about the biological interactions of pest, crop, and physical environment.

Therefore, the questions that need to be asked include who is managing the activities and why; what organisational processes these activities depend on, and so on. Actors include the Producers (farmers) and Local Government (who depend on farmers for Integrated Production, and obtain this by registering trademarks for the farmers and funding an advisory service for them, thus forming a feedback loop on the SD diagram).

The apple codling moth can be controlled before egglaying by preventive actions such as Pheromone Trapping (PT) to lower mating; or afterwards by periodic assessment of the degree of infection and the choice of a remedy, known as a disease management plan. So, PT needs to be planned using maps of the geometry of orchards and infection sources, diffusion barriers, etc. Planning needs models of the 'evolution' (during a season) of the pest. The solution – a Decision Support System (DSS) -- clearly therefore needed both accurate maps and detailed advice based on the biology and the geography. In the i* SR model, the DSS takes over several of the dependencies (and tasks) of the advisor.

Like the last speaker, Perini simplified the i* notation, using the familiar notation of simple arrows to mean OR-decomposition of goals, and arrows linked by a curved line to mean ANDing. Practitioners, in other words, even in research institutions, seem to find i*'s SR concept useful, but too tricky in its original notation. Perini also pragmatically used softgoal to handle goals that were unclear at the start, even if they weren't conventionally 'soft'.

Designing an effective DSS for a problem in this domain requires a deep understanding of its organizational dimension to handle all the strategic dependencies between the domain stakeholders. The final output of the project included a prototype of a GIS-based system.

The organisational model helped with communication between analysts and stakeholders (including system users). It allowed analysts to reason about process re-engineering.

Professor **Oscar Pastor**, Valencia University of Technology, Spain spoke about Some Lessons Learned from using i* Modelling in Practice.

Over the last year he has been using i* for organizational modeling in local software production companies. The idea is to generate conceptual models from requirements models, and then to generate formal specifications. But the 'why' was missing. That role is now taken by i* for early requirements. After that, model-driven development takes over.

Pastor has studied i* in three cases: conference workshop management; golf tournaments; and car rental. Three groups of people – experts in OO modelling (but not i*); requirements modellers (also not i*); and i* experts (PhD students) took part. The experts taught the other 2 groups the basics of i*.

i* modelling was very useful for enterprise modelling, yielding knowledge of bottlenecks (actors with too many dependencies) and vulnerabilities (actors depending on other actors for all their goals).

But there are some worrying problems.

Repeatability: different analysts create different i* models for the same situation: some use tasks, some goals, some resources (between the same pair of actors). These would lead to scenario steps within a use case, a new use case, and a new class in a sequence diagram respectively.

Scalability also looks problematic: models quickly get large and complicated.

Encapsulation is tricky; models are richly interlinked. How can you (de)compose i* models?

Understandability is affected by all the above issues.

Traceability between i* requirements models and later models (behavioural, etc) is awkward. You can easily add i* elements, even if they didn't come from a recognisable source in an earlier model.

The experience has been contradictory: on the one hand all the practitioners involved in the project have recognized the benefits of applying organizational modelling techniques, and i* has been seen as a rich solution for that. i* is ideal for describing business services: customers depend for <some service> on the providing business. The business and customers have goals (profit, growth, satisfaction, …), which i* can describe well.

On the other hand, the resulting complexity has been difficult to deal with, especially when applying i* to larger systems, and some concepts have shown to be easily misinterpreted. It is risky to present i* models to third parties.

A questioner suggested that Pastor had completely sidestepped the intentionality of i*. He replied that this was true in the functional modelling done so far; it would not be true of future non-functional work.

Professor **Neil Maiden** and Dr **Sara Jones**, City University, London spoke on Model-Driven Requirements Engineering with i*.

Over the last 4 years City University has applied its RESCUE requirements process to specify the requirements for several European air traffic management systems. City people trained third party analysts in NATS and its french counterpart, and those analysts created i* models.

The i* approach is a key element of RESCUE, used to model the goals of, and dependencies between, software and human actors in the complex socio-technical systems found in air traffic management. It is supported by the REDEPEND tool for i* system modelling. RESCUE supports 4 concurrent work streams: human activity modelling; i* system modelling; use case and scenario analysis; and requirements management (with ReqPro, which "we pushed almost as far as it could go"). These of course interact, raising the question of when and how the models should influence each other. Other questions include how to scale i* up to large socio-technical problems like air traffic control. The approach was action research rather than case study.

ATC is a large scheduling problem as there is a limited number of runways (specially at Heathrow). The departure manager (DMAN) system is used by 7 or 8 people in the control tower at Heathrow; they all have different roles and depend on each other. These were observed and described as 15 as-is scenarios (human activity modelling). These in turn were made into 15 use cases, each with 13 normal course actions and 3 variations on average. While that was going on, i* SD and SR models were made with 15 actors and 46 dependencies. The team developed a custom add-on to the graphics tool VISIO, REDEPEND, to draw the i* diagrams. The french team made a model with 7 actors and 103 model elements, making a large SR model which couldn't be merged with the english models. So most of the work was eventually done with SD models.

The project also wrote some local rules for using i*. You mustn't just draw an SD diagram from scratch! Much better to start from an extended Context Model, which is allowed to have actors with interactions outside the system boundary. This is a much better startpoint than an SD model. It's also valuable to make a table to document dependencies, with 3 columns for A, depends on (type), B. Then the diagram is easy to draw.

Task dependencies are hard for people even in research institutions! We stuck to goal-softgoal dependencies only. And having a simple VISIO-based tool meant that everybody could use i* on their desktops.

The project had 5 synchronisation stages between the workstreams, each one being to make decisions, starting with boundaries (from the context modelling). Synchronisation checks included seeing that actors, resources, goals, actions, variations and differences due to contextual features in all activity models should also appear in the relevant use case descriptions. The checks were in other words simple, and were applied manually "over a crazy 8 days". Since then, tools have been built to cross-check the models automatically. Analysts can then fix the discovered problems efficiently. 130 or so issues were discovered during the checks.

For instance, if A depends on B for a resource, it implies that B has the resource before A has it. But if in the use case A is using the resource before B, there is an inconsistency.

i* dependencies also revealed gaps in the use cases, such as steps to plan aircraft departure sequences. A whole new use case was also discovered during cross-checking. Also, use case dependencies can be identified. Analysis of the i* models thus yielded more complete and more consistent use cases, scenarios, and requirements.

David Bush (of NATS) said that the i* modellers were just normal people (as opposed to researchers) [laughter].

Richard Veryard asked if the project used the intentionality aspects of i*. Maiden answered that it depended what you wanted to do; if you didn't want to achieve a goal then it was clearly outside your system boundary. But knowing the goal existed was helpful; it afforded you the option of thinking about how that goal could be satisfied. Different conceptual tools help you to think in different ways. Scenario walkthroughs generate numerous functional requirements. i* models are good at hunting down non-functional requirements.

The day ended with a panel session moderated by Ian Alexander.

Charles Symons asked what Kolp had used i* for, what his goals were. Was it a patient management system, ie for the benefit of the consumer, or a bed management system, ie for the provider? Kolp was non-committal. Yu said that i* had in the past been used from the provider's point of view. Richard Veryard said that a 'system of systems' solution would cover both aspects, and how to fit them together. Pastor said that context was important. i* was good for modelling organisations. European hospitals had different goals from American ones. It would make sense to make several similar models to explore different contexts. Yu said that the different stakeholders had (some) common interests.

Another questioner said they had the impression from the talks that i* was good on large projects that used a 'waterfall' process. Could it be used iteratively in eg a (Barry Boehm-like) spiral model? Maiden replied that it was, and his City team had experience of that, eg using i* iteratively in its work on submarines. Kolp said that TROPOS helped with using i* in the sort of process needed in the real world to feed modelling results into requirements iteratively. Pastor said that there were three levels -- organisational modelling, conceptual modelling, and software at the bottom. Their development can run in parallel -- that may not be the best approach, but it is possible.

The final question from the floor was to the whole panel: what is the most important use of i*?

Perini said it was an intuitive graphical way to represent concepts, and to show users the models and explain how dependencies can be changed using the tool.

Yu said you get people thinking WHY; the modelling method is helping with the dialogue.

Pastor said that conventional modelling asks WHAT and HOW. This approach also asks and answers

WHY. i* is the best option we currently have for the why-question.

Maiden said i* was an effective technique for socio-technical systems. It provides a much richer way of exploring system boundaries than use case or conventional context modelling; you don't have to pre-judge where the boundaries were, but simply go right ahead identifying dependencies and then try putting the boundaries in different places.

Ian Alexander thanked all the speakers on behalf of the RESG. We had depended on them for a wonderful day.

© Ljerka Beus-Dukic 2005

Goal-Oriented Requirements Engineering with i*: Suitable for Service-Oriented Architecture?

by Richard Veryard, an independent consultant.

Last week I attended a seminar organized by the BCS RESG. Modelling your System Goals: The i* approach. The main speaker was Eric Yu (University of Toronto), the original developer of i*. The remaining speakers reported their experiences using and extending i* in a range of projects.

The i* approach is a methodology for goal-oriented requirements engineering (GORE). Professor Yu characterized the distinct nature of Goal-Oriented RE as shown in the table.

i* is particularly focused on so-called "early stage" requirements engineering - before you know what "the system" is going to be. It involves two key models: strategic dependency models and strategic rationale models.

Conventional RE	**Goal Oriented RE**
Describes how things work (AS-IS) and how they should work (AS-SHOULD-BE).	Describes why things work (or should work) that way.
Models of behaviour.	Models of intention and rationale.

The strategic dependency model identifies goal dependencies between actors. For example, a CarOwner actor depends on "CarBeRepaired", and this can be fulfilled by the BodyShop actor. Put crudely, a goal dependency consists of a pair: {"I want" (on one side), "I can" (on the other)}.

i* distinguishes between two different kinds of goal dependencies: hard goals and soft goals. Although this distinction was explained in terms of precision, this seems unsatisfactory to me, since surely precision is itself a matter of degree and timing and negotiation. However, the soft goals that were used as examples seemed to me to possess some other interesting features, which may be relevant to making this distinction clearer:

- Firstly, the soft goals were generally not amenable to precise specification by the consumer of the goal.
- Secondly, there is a sense that what counts as satisfaction of the goal is context-dependent and shifts over time.
- Thirdly, the perceived outcome would generally affect the relationship of trust between the consumer and the provider, and thus the future behaviour of the consumer towards the provider.

This led me to an alternative way of formulating the distinction - according to who owns the goal. In some cases, the consumer determines and decomposes the problem, and then delegates the solution. I think this produces a hard goal that is (or should be) relatively easy to specify. In other cases, the consumer delegates the problem (formulation of) as well as the solution (provision of). I think this would produce what i* calls a soft goal.

The strategic rationale model identifies causal connections between goals and subgoals for a given actor. Note that the behaviour of an actor may depend on perceived causal connections and associations, rather than actual ones. So if we are going to model rationale properly, we need to include some representation of the actor's beliefs. However, last week's i* presentation did not introduce any notation for belief.

SOA interpretation

Service-orientation (Service-Oriented Architecture or SOA) is an approach to the composition of complex systems from independent or semi-independent services. This approach can be applied at several levels:

- software engineering (where the services may be rendered using web service protocols such as WSDL/UDDI, often intended for broad reuse across organizational boundaries);
- systems engineering (where the design objective is often to produce highly complex federated systems of systems, without a central requirements/design authority), and
- business strategy (where business viability increasingly depends on the capacity to deliver added value services with sufficient differentiation and integration).

Much of the i* material presented was pre-SOA in outlook, and there were some limiting assumptions about the nature of the systems being built. However, I was looking to push i* beyond these assumptions. What I wanted to explore was the possibility of using i* to design services and service-oriented solutions.

There appears to be a very straightforward mapping from goal dependencies on the strategic dependency model to business services. And these requirements are supported by an understanding of the rationale of each actor. The reason for this is that we want to design business services in terms of added value, and this seems to rely on our having some notion of value from the perspective of the service consumer.

Another SOA-related motive for modelling dependency and rationale is to analyse compliance (including so-called non-functional requirements). For example, we may identify some system vulnerability, and then identify a reciprocal dependency that provides some enforcement mechanism. We can then look at the system dynamics within a collaboration, to determine the adequacy of this mechanism, as well as any possible side-effects.

In general terms, I am convinced that some modelling of intentions and outcomes is an important aspect of modelling for SOA. In the Boxer-Cohen Triple-Articulation approach to Asymmetric Design, this is known as the Deontic Articulation (The dictionary says deontic means ‘Of, relating to, or concerning duties or obligations: deontic logic’. Ed.). i* is much simpler, and widely practised - but is it powerful enough?

Complexities, difficulties and future opportunities

Modelling the intentions of people and other agents

There are difficulties involved in modelling the intentions of both organizations and machines, and these difficulties cause some people to say that it doesn't make sense to attribute intentions to either organizations or machines - only to people. I don't agree, for two reasons. The first is that many of these difficulties also apply when we are trying to understand the intentions of people, since people often have confused or concealed intentions. The second is that there are established techniques for tackling these difficulties, which help us to understand the intentions of all kinds of system, at different levels. The third is that we simply cannot make sense of business strategy and organization at a reasonable level of abstraction without somehow talking about organization goals.

Goal mismatch

In practice, there is unlikely to be a neat and exact match between "I want" and "I can". There may be both pragmatic mismatch (crudely: what the provider does is not exactly what the consumer wants) and semantic mismatch (crudely: what the consumer understands us not what the provider understands). This is a form of impedance or asymmetry.

For example, a car driver's real goal may be to have a reliable car with low total maintenance cost. But no service provider offers exactly this service. So the car driver has to compose something that approximates to his real goal, using a combination of car maintenance services and car warranty (insurance) services.

Within i*, the strategic dependency assumes an exact match between the consumer's view of the goal and the provider's view of the goal. To the extent that there is some unfulfilled remnant, this is analysed within the strategic rationale of that actor. But it is not clear to me how this approach would support architectural reasoning about the unfulfilled remnants and the strategic opportunities for developing new added-value services.

Deconfliction

Given that a complex situation contains considerable goal conflict, we need a systematic way of resolving goal conflict. Deconfliction means organizing operations in a way that minimizes the potential risk of interference and internal conflict. Conversely, we need ways of resolving goal synergy.

Note that from a security perspective, goal synergy between independent actors may not be a good thing - especially where it indicates opportunities for collusion and fraud.

Variation

A robust service economy typically needs to accommodate considerable variation in the goals and rationale of the actors. So instead of modelling a single standard strategic rationale, we need to understand the nature of the variation between different rationales, and the implications of this for producing robust and agile systems.

For example, a healthcare system makes some assumptions about the rationale of a patient. But a patient that happens to be a professional athlete (with particular concerns about stamina and performance) will have a very different strategic rationale in relation to healthcare, as compared to a patient that happens to be a healthcare professional (with above average exposure to infection and other risk).

Dynamic and recursive rationale modelling

The causal relationships between goals and subgoals are subject to dynamic effects. For example, many processes operate in terms of surrogate goals - outcomes that are valued not for their own sake but because they are correlated with some real goal. The surrogate goals are available for measurement before the real goals, and may therefore provide useful predictive metrics. (See discussion of Surrogate EndPoints in relation to SOA Pharma.) This means that cause-effect modelling may need to include system dynamics such as delay and oscillation.

There is also a problem that the rationale of one actor may depend on that actor's beliefs about the rationale of another actor. For example, Professor Yu presented an example from insurance, where reorganizing the strategic dependency model could align the insurance broker with the interests of the customer (instead of the broker being aligned with the interests of the insurance provider). In this situation, what the insurance customer demands from the insurance broker depends crucially on the customer's belief as to whose side the broker is on. But that in turn may depend on the insurance broker's beliefs about the customer's acting in "good faith", and about the future commercial tactics of the insurance company. So the whole thing becomes recursive, rather like R.D. Laing's Knots.

Summary

I think the overall approach of i* is extremely interesting for service-oriented architecture, and is broadly consistent with the general approach to business modelling for SOA that we have been developing.

© Richard Veryard 2005

RQ readers may be interested in Richard's website which discusses many SOA issues: http://www.veryard.com/so/soapbox.htm

An Audience with David Parnas

25 May 2005, Imperial College, London

This was a rare chance to hear one of the big names in software engineering. David Parnas is an early pioneer - an icon to many - of software engineering who developed the concept of modular design, the foundation of object-oriented systems today. His double dictum of high cohesion within modules and loose coupling between modules is part of the bedrock on which encapsulation as a design technique is built.

Parnas earned his Ph.D. at Carnegie Mellon University, and worked there as a professor for many years. He also taught at the University of North Carolina, the Technische Hochschule Darmstadt, and the University of Victoria. He then went to McMaster University in Hamilton, Ontario, Canada. He currently works at the University of Limerick in Limerick, Ireland. He earned a professional engineering license in Canada and was one of the first to apply traditional engineering principles to software design.

Distinguished Speaker Lecture: Prof. David Parnas

Wednesday 25th May 2005, Imperial College, London

A Systematic Approach to Requirements Documentation

Parnas began by saying that he absolutely hated the term Requirements Engineering, as it isn't a branch like Mechanical Engineering but something that every engineer has to do.

He said that he liked at the start of a meeting to address the audience in their own language. He apologised that as he'd lived a long time in New York he'd have to use his own language instead (which sounded a lot like English actually).

He said he was a software voyeur, someone who looks at other people's dirty systems. Clearly the lecture was going to be packed with spicy anecdotes. Here are a few more.

Every engineering course begins with some propaganda about the responsibilities of an engineer. He remembered some from his: to make sure that products are 'fit for use'; not to do what we're told but to do what is needed.

Requirements documents have 2 completely different audiences, that Parnas calls 'users' and 'implementors'. They have to be readable by both.

- Good requirements have to be corrected by the users. He is proud that his A7 aircraft specification was fiercely reviewed by the pilots: they found 500 errors. That was good, as it meant they read it in intimate detail.
- Good requirements have to pass the coffee-stain test: do you find scruffy stained copies of the requirements on the programmers' desks? If not, you've written pious generalities. Everybody thinks it would be nice if the system was 'easy to use'; so the rule is, if nobody ever criticises a requirement, he crosses it out!

He says he cut down a Philips specification to 10% of its original length by applying this rule. By the way, he quipped, political speeches usually disappear completely if so processed. All the errors were in the cut-down document (ie, the other 90% was indeed just fluff); the reviewers loved it as it was readable and focussed on the actual requirements.

Parnas' heart pacemaker has a ridiculous feature: the resting heart rate can only be set in multiples of 10 beats per minute. If your resting rate is, say, 75, you have to choose between too low and too high. Nobody had reviewed the requirement with doctors; it was just a silly guess by a programmer. Consultation and review are vital parts of the process.

For instance, fighter planes have 2 altimeters; if one is broken or seems to be giving unreasonable results, the other takes over. But what if both fail? Programmer asks pilot: what is the average altitude? Answer, 37500 feet. Result: if both altimeters fail, the display changes to 37500, and the pilot's manual now teaches that if the display goes to this for more than a few seconds, the pilot must pull up (to avoid flying into the ground)! Parnas said, what's this with 37500 feet? Why don't we write the requirement as to pull up? They showed him the pilot's manual, which had fossilised the original error.

Software that is 'almost right' is wrong. Using 'Undefined Modelling Languages' (sic) is like walking into a quicksand. Even UML advocates admit they don't have a formal definition of model syntax 'yet'. Parnas accosted Ivar Jacobson with the inconsistencies of UML, and Jacobson knew them all! But he said if he admitted it, the enterprise would founder. UML confuses engineering with commerce.

So, what are the **Acceptance Criteria for 'Descriptions' of software**? They must

1. be easier to understand than the code

2. not restrict the solution unnecessarily
3. enable testing and proof to be automated (eventually).

Obviously this calls for something mathematical (#3) but easy to read (#1) and in the problem domain (#2). Is that possible? Parnas says it is.

Formal methods like Z or VDM are no good, as they are harder than code; people prefer to read the code if they want to see what it does.

Requirements documentation should

1. not be a sales pitch!
2. serve as a reference during development
3. prevent programmers from taking unilateral decisions that should be made collectively
4. be used for design reviews
5. be used to generate test cases and to evaluate test results
6. be kept up to date during maintenance
7. be modified to specify revisions (and variants) for different contracts.

Writing requirements this good takes time, but it's worth it.

Bashar Nuseibeh wondered if contractual requirements could avoid turning into bloated legal-speak. Parnas said yes, you just stapled the engineering requirements to the legal stuff, and on arrival at the contractor the two bits were at once separated again.

The goals for requirements documents are to:

1. decide what to build before building it
2. provide an organised reference for the developers (including Quality Assurance)
3. allow for staff turnover.

The old 2-variable model of (real-time control – Parnas doesn't say so, but this is the domain he's thinking about) hardware and software assumes a system has inputs and outputs, and outputs = f(inputs). Unfortunately the function f can get very hard to write, and harder to review.

Parnas' 4-variable model says there are not only inputs and outputs, but there are I/O devices either side of those (outside the inputs and outputs!), and those I/O devices receive Monitored variables and manage Controlled variables (both of which are in the world):

M-var => I/O device => inputs => software => outputs => I/O device => C-vars

So the first thing Parnas looks for is a list of all the Monitored variables and all the Controlled variables.

For instance (another anecdote), you don't want your plane to drop a bomb while it's on the ground. It should be a requirement that bombs cannot be released (Controlled variable) while there is weight on the wheels (Monitored variable): of course that wasn't anywhere in the specifications.

Just creating such a list and defining what each Monitored and Controlled variable is, is a big task.

As Michael Jackson says, we need a precise and complete list of designations, references to things in the world outside. We can't make good software with tunnel vision, looking only at things inside the computer.

This led Parnas to frame two (mathematical) rules:

1. the Requirements domain must span the Natural domain, ie if something can happen in Nature, you must know what the system is Required to do about it – all events must be handled, or your system will fail when the first unhandled event occurs. In set notation:

   $\mathrm{domain(REQ)} \supseteq \mathrm{domain(NAT)}$.

2. what is Required must be feasible with respect to Nature; you mustn't ask the system to violate the laws of physics. In set notation, we are interested in the intersection of the two sets:

   $\mathrm{domain(REQ} \cap \mathrm{NAT)} = (\mathrm{domain(REQ)} \cap \mathrm{domain(NAT)})$

These rules let you check for completeness and consistency (but not, alas, for correctness!). REQ is of course about the Controlled variables, the things we require to be made as we want; NAT is about the Monitored variables, the things we need to watch.

The procedure for documenting the requirements is therefore

- list the Controlled variables;
- list the Monitored variables, defining their ranges (and hence, the Natural domain);
- define the Required value of each Controlled variable at time t as a function of the values of the Monitored variables at time t or earlier.

When the list of variables is complete, and the functions cover all possible values of the Monitored variables, the document is complete.

As long as each Controlled variable is mentioned in only one function, the document must be consistent. It's all elegantly simple in concept.

One fly in the ointment is the question of Modes: are we in real mode, training mode, software update mode, or what? A mode is not a state but a whole class of states with their allowed state transitions. For instance, in training mode you may have all the display screens and push-buttons that you get in real mode, but different results. It can get confusing, as the same screens or very similar ones may be used in different modes. A pilot (ah, another anecdote!) had better know whether pressing the red button on the stick means 'drop the bomb now' or 'reset the plane's position to the co-ordinates of the known landmark' (eg as you fly over The White House).

A question mark hovers over the issue of documenting Exceptions, Undesired Events. Much software handles exceptions terribly badly. Parnas gave the example of the error message he gets when his laptop isn't correctly plugged into the network. It says the web address doesn't exist or hasn't been typed in properly. He doesn't know how often he's retyped something when he should have checked his cable! A problem is that people like requirements to sound bright and cheerful, to talk about all the good and positive things the system is going to do; people don't like talking about Exceptions. What is undesired is purely subjective, so in theory you don't need to handle some subset of events as special cases. In practice though, people tend to overlook them, so it's probably best to keep the Exceptions section in.

Parnas was asked if it was true that Exceptions made up 80% of the requirements. He agreed. Also there is 'no practical upper bound on the number of things that can go wrong', so you always have to decide where to stop worrying.

He also advocates having a section of Assumptions and Expected Changes. He suggests 2 complementary lists – likely changes, and fundamental properties. This enables programmers to design for change, to a reasonable extent. You really have to stretch your imagination to think of likely changes. People won't tell you what will change; but they will tell you if you're wrong if you propose something, so you have to be brave and stick your neck out.

Assumptions are hard to pin down, too; what people tell you must be so, based on their knowledge of old systems, is probably wrong given new technology. (See the book review below of James Dewar's *Assumption-Based Planning*, which suggests 9 techniques for surfacing hidden assumptions.)

Still, a spartan system is better than nothing. If your system does the vital things (like, if something's wrong, get the plane home, any way you can) then it'll be workable.

The requirements for acceptable software are the results to be output from the software functions applied to the input data – obviously you can say that mathematically (as a functional expression, or better but less readably as a logical relationship). The practical outworking of this is that you can specify sofware using tables.

Tables

- divide and conquer complexity, creating a 2-dimensional structure that presents a set of simple expressions
- have already been 'parsed' for you: you only have to read a part of the table to start using it
- can be checked for completeness and consistency
- have precise meaning ('they are every bit as formal as mathematics')
- can be interpreted and evaluated automatically
- are readable
- can be used in many contexts.

Software is trickier to specify than traditional engineering as there are so many discontinuities. Tables work well because each discontinuous chunk can be dealt with in a cell or row of a table. Continuous functions are equally easy, you just write the functional expression that applies in a particular case into the relevant table cell.

'I see tables as mathematics with a 2-D presentation', said Parnas. There's no theoretical difference, said a colleague (he reported). No, there's just a practical one, he replied: people can read the tables.

Functional approaches are superior to writing Preconditions and Postconditions, as they are simpler, and complete. The Pre- and Post-conditions aren't separate things but two parts of the same model.

A convenient notation avoiding explicit set theory (but equivalent to it) is to define @ some function of x to mean the event of x becoming false; and $state$ to mean a state. Then you can just write the rule that an aircraft's undercarriage must not be retracted until the wheels are off the ground as

@T(altitude = 0) when (wheels = $retracted$)

Such case-based reasoning is simply essential; it is much easier to get right than trying to write universal rules that cover all possible cases. Breaking things down into tables is powerful because you soon discover cases that nobody thought about.

Why aren't these tools all in a compiler? Because most of these black-box specifications are read by people who never read code.

When people just try to write code they usually get it wrong. When they write tables they usually get it right. This is Parnas' answer to the XP (Extreme Programming, Agile Methods) people: they are responding to the real problem of 'write-only' documents. They want to go straight to defining test cases. But taking time to define tables enables you to identify the test cases directly: each cell is a test case.

Is this limited to real-time control software, or is it a completely general concept? Answers on a postcard to the Editor, please. Certainly if we're dealing with business processes, etc, the "I/O device" must be metaphorical; a human manager is the controller based on the information read off a screen, etc. Parnas' own answer (your Editor asked him) is that while it works with real-time, that's just an additional constraint; remove it, and the method still works. He's done it with Management Information Systems, where the inputs and outputs are numbers and pieces of text; the relationships between these can still be specified functionally.

Parnas held everyone's attention, fascinated, amused, possibly provoked, possibly stimulated to go out and specify with tables, for all of a 3-hour tutorial.

© Ian Alexander 2005

RE-Papers

Seven Principles of Market-Driven Requirements Engineering

By Björn Regnell, Software Engineering Research Group, Dept. of Communication Systems, Lund University,

http://serg.telecom.lth.se

How should you go about designing your requirements engineering process when focusing on selling software-intensive products to many customers? More and more software (either embedded or packaged) is produced for an open marketplace rather than to one specific customer and it is getting increasingly important that both industrial practice and academic research address the special requirements engineering challenges in this context. In market-driven software development the costs of generic products are divided among many buyers on open markets and the potential profit is rewarded to the producer. This situation is clearly different from bespoke requirements engineering

In a survey on market-driven requirements engineering [1], a number of challenges specific to market-driven requirements engineering were identified:

- **Balancing market pull and technology push**. It is necessary to find a good trade-off between requirements corresponding to perceived user needs and new, inventive ones that may provide a competitive advantage through groundbreaking technology. Finding a good balance between technology-driven and needs-driven requirements may be a delicate challenge.
- **Chasm between marketing and development**. In some companies it can be observed that there is a gap between marketing and developers concerning the views on requirements engineering. Better communication and collaboration between these groups are needed, in order to increase the requirements quality and thereby the quality of the final product.
- **Organizational instability and market turbulence**. Companies without a defined process take a significant risk if key persons leave the organization, since they lack the necessary documentation and structure. In times of downsizing or rapid expansion it is very difficult to install a repeatable process.
- **Simple tools for basic needs**. Some companies requested simple and easy-to-use techniques for basic activities. For these companies it was a challenge to find solutions that are not too complex.
- **Requirements dependencies**. Dependencies among requirements make release planning difficult. Some companies treat dependencies in a basic way by bundling related requirements, but efficient ways of managing at least the most important dependencies are needed.
- **Cost-value-estimation and release planning**. Release planning rely on accurate estimates; underestimation of cost may result in an exceeded deadline while over-estimation of cost may exclude valuable requirements; over- or underestimation of value may result in a product that is badly aligned with actual market needs and thus make the development investment a losing business.
- **Overloaded Requirements Management**. Requirements suggestions from developers and customers are essential, but it is a challenge to prevent the requirements repository from being flooded with requirements and how to maintain throughput at times when the number of arriving requirements peak.

Some of these challenges have been addressed in our recent research within requirements engineering, and during these investigations we found that a number of practical principles emerged based on evidence or indications in industrial case studies relevant to the market-driven context. These principles (although not yet completely investigated with respect to their generality) can be taken as hints on what might be good process elements to consider when designing a successful market-driven requirements engineering process.

Principle 1:
Continuous elicitation with smart screening

As software products often are released at regular intervals the development is continuously enhancing the product to meet increasing market needs. Interaction with customers and users through sales, support and trouble reports is increasing as the number of sold units increases. Competitor intelligence is getting increasingly important as the competition on the market gets harder. All this calls for an organisational function (rather than a project phase) that can cope with the continuous elicitation of important information that is input to the decision-making part of the requirements engineering process.

Furthermore, as the amount of customer increases and the specialisation of the product variants (perhaps based on a product-line approach) increases, it seems inevitable that the development organisation faces the risk of information overload. Some of the information regards requirements that never will be implemented, but it still takes time to process. It is likely that the requirements engineering function will need a smart screening function where requirements that are estimated to be out of scope are cancelled before too much effort is spent on them.

**Principle 2:
Aligned multi-level information management**

The incoming candidate requirements are often informal with varying quality. They are likely to be both incomplete and ambiguous and thus difficult to use for predictions and estimations of development effort and market value. It is however not realistic make all incoming requirements perfectly specified and we have to live with incomplete and unambiguous models and develop efficient heuristics to tackle the requirements information quality problem. We have seen that requirements engineers build models on different abstraction levels with varying degrees of structure and formality. The information elements include e.g. *high-level business goals* that encompass the business strategy of the developing organisation as well as the business logic of the customers; *feature models* providing a view of the product functionality that makes sense from an external view point (in the eye of the users and marketing); *use cases* that explain in principle how the user in co-operation with the system reaches their goals; detailed and vivid *scenarios* that can help explain the actual reality of users problems, detailed *design-level requirements* often related to the user interface metaphors of the system or induced by the fact that the system must interact and integrate with existing systems. There is often a distinction between requirements as customers or users have put them and requirements as the developing organisation has understood and specified them in relation to their own business strategy and product architecture. It seems crucial to manage information on such varying levels of abstraction and at the same time try to align the different levels (as much as possible within the given resource restrictions) in such a way that they can be used in concordance.

**Principle 3:
Similarity and interdependency tracking**

In order to support principle 2, it looks crucial to devote a significant amount of time to analyse the multi-level requirements with respect to similarity and interdependency. Duplicates are not obvious to spot, but spending time on the same thing twice is a waste and it is worth while to try to eliminate duplicates as early as possible. Many similar requirements coming in may be an indication that many customers want the same thing, which in turn is valuable information when trying to find profitable new features to implement. It is also important to maintain links between the requirements on different levels, e.g. between the requirements as stated from the customers and users viewpoints and the detailed requirement that are allocated to actual development projects. Complex interdependencies among requirements makes the predictions of market value and development effort even more difficult and many practitioners seem to try to spot a limited number of crucial interdependencies that are managed with well thought through heuristics. The potential number of links among requirements that could be traced is immense and a successful organisation must learn which relationships are justifiable to maintain. A promising set of techniques to support spotting and tracking inter-relationships can be found in the areas of statistical linguistics, text analysis and information retrieval [2].

**Principle 4:
Accurate and recurring prioritisation**

The market evolves over time, as well as the software products and the developing organisation. Requirements rankings that were set at the beginning of a release cycle are often reshuffled as elicitation provide new data or customers change their behaviour. Perhaps the most critical capability of a market-driven software development organisation is to outperform the competitors in the accuracy of value and effort predictions that are the basis for prioritisation. The predictions are uncertain and based on subjective views and experiences from previous releases, but they do not need to be perfect as long as the decisions on what requirements to select are more effective than the selections made by competitors.

There are many ways of doing prioritisation and each organisation need to make a well-informed decision of what fits best in a given situation. Evaluations of prioritisation methods indicate that different methods are suitable in different contexts and that the methods have different pros and cons.

**Principle 5:
Adaptive release planning**

The principle of accurate and recurring prioritisation is a fundamental basis for the critical release planning activity where requirements are selected and planned for implementation in a certain release of the software product. Release planning is not just a simple optimisation task of packing the product with a set of features that amount to a given cost and meeting the delivery time point. Release planning is an inherently complex task of finding a set of requirements that is well-enough aligned with many disparate aspects such as short-term marketing constraints ("we must show something cool on that exhibition to get media attention") and long-term architectural evolution strategy ("we need to do a costly rebuild of our internal data storage mechanism to meet future performance requirements").

Furthermore, as the effort and value predictions that form the priorities governing the release planning is uncertain the release planning needs to take into account the technical and commercial risks that results in unintentional over- and under estimation of cost and value. Risk buffers and protections against over-allocation of available resources need to be built into the release plans. As soon as unforeseen killer requirements are revealed the release plan should be updated and less rewarding requirements postponed in a structured way.

Principle 6:
Process capability monitoring

There seem to be an obvious risk of information overloading hampering throughput in the requirements management process. It is important to design a measurement scheme that can help monitor the throughput of the process both with respect to quantity and quality. How many requirements are coming in per week and how does this figure vary over time? How much time do we spend on each requirement in different phases of requirements refinement? How many mistakes are made based on insufficient requirements specification quality? In order to be able to spot overloading and bottle-necks we need to have quantitative measurements of the process capability combined with qualitative surveys of the views of the involved engineers on how things are going. It is not likely that high-level management can take the right decisions regarding when to increase the number of resources on requirements engineering, if there is no hard data providing evidence for the need and its benefits compared to rambling coding, which seem to be the factual behaviour of many software projects.

Principle 7:
Structured post-release learning

Some things that we wish to know about the requirements engineering process cannot be determined until the product has been out on the market long enough to receive the merciless feedback of its customers and users. How many requirements were incorrectly selected for implementation? How many requirements were incorrectly rejected? Which features turned out to be the most lucrative and the most unprofitable?

We need to go back and put our decisions under scrutiny through a structured post-release analysis in order to maximise the learning. One of the most valuable sources for increasing our situated understanding of market-driven requirements engineering is at hand in our own organisation's experience. In general we would like to know the answer to this critical question: Why were these exceptionally good or bad decisions made? If we knew the answer to this question, we would also have a handle on what to change in order to improve requirements process capability.

Perhaps the most successful software companies will be those that succeed in developing effective and efficient strategies for continuous requirements engineering process improvement, enabling them to endurably stay ahead of competition.

Conclusions

These principles address some of the hard challenges that market-driven software developing organisations face. The overload challenge and the release planning problem are partly addressed through smart screening and systematic prioritisation. Available prioritisation methods also give a handle on the difficult trade-offs within and between market needs and technological opportunities. The issue of requirements dependencies is partly tackled by similarity analysis to support requirements bundling and by selecting only the most important relationships to track. Organisational issues can be further understood in context if structured post-release learning is applied in continuous process improvement efforts. Communication chasms within an organisation can to some extent be over-bridged by a systematic alignment of multi-level information management enabling both marketing and development to connect their respective models to each other. We have a number of starting points, but it is obvious that we need to struggle further with all these hard challenges and improve our processes, methods and technologies as the frontier of RE capability competition moves ahead. And for us academic researchers, the challenge of providing more practical, simple-to-use tools appropriate for the basic needs of market-driven requirements engineering deserves indeed to receive our devoted attention.

© Björn Regnell 2005

References

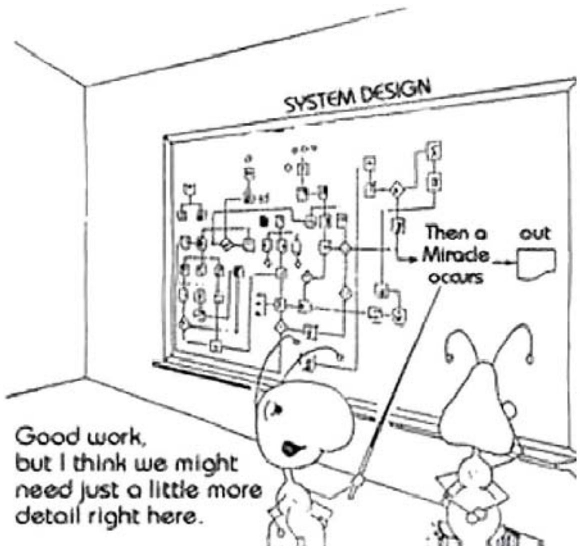
[1] Karlsson L., Dahlstedt Å. G., Natt och Dag J., Regnell B., Persson A., "Challenges in Market-Driven Requirements Engineering - an Industrial Interview Study", *8th International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'02), September 9-10th, Essen, Germany, 2002, pp. 37-49.

[2] Natt och Dag J., Gervasi V., Brinkkemper J., Regnell B., "A Linguistic-Engineering Approach to Large-Scale Requirements Management", *IEEE Software*, Vol. 22:1, January/February 2005, pp. 32-39.

More references can be found under 'Publications' at:
http://www.telecom.lth.se/Personal/bjornr/

New Approaches to Specifying Clear Testable System Requirements

David Gelperin, ClearSpecs Enterprises


SYSTEM DESIGN
Then a Miracle occurs
out
Good work, but I think we might need just a little more detail right here.

"Then a Miracle occurs" is a label that fits software development in many project workflows. Since fuzzy concepts from customers must be transformed into acceptable precision (code, data, and tests), analysts, developers, and testers are the designated miracle-workers. All too often, however, no miracle occurs.

Poor definitions of "acceptable" are a major cause of unacceptable systems. Many system specifications contain pages and pages of text – sometimes supplemented with tables, diagrams, and a natural language glossary.

The difficulties in creating unambiguous and precise specs using mainly natural language have long been recognized. Formal specification languages have been proposed as a solution, but such specs are hard to understand.

The long-standing need for a comprehensive framework of clear testable specification patterns has been satisfied by the ClearSpecs framework.

This framework:

- contains 18 patterns that are easily understood by all project stakeholders
- supplements, rather than replaces, currently effective specification practices
- covers each aspect of requirements specification for systems and components with the most precise but understandable patterns available
- supports a range of approaches to requirements development from agile to traditional
- enables automatic design of high-quality acceptance tests

The ClearSpecs framework supplements 5 natural language descriptions with 13 precise patterns. Some of the precise patterns are familiar, such as decision tables, state transition tables, and entity (class) specs. Others are new or unfamiliar in system specification, such as precise use cases, quality specs, and derived conditions. Extensive use of the precise patterns results in clear testable specifications.

Requirements teams, using only traditional specification patterns at the start of a project, often uncover significant misunderstandings when precise patterns are introduced. In addition, they find critical "subsurface" issues that need immediate attention.

Details about the newer specification patterns can be found among the technical papers available at http://www.clearspecs.com.

© David Gelperin 2005

Category	*Spec. Pattern*	*Purpose*
Overview	1. Background	Information that connects development work to organizational needs, decisions, and history
	2. Features	Natural language descriptions of the system features to be developed or modified
Usage Models	1. User Stories	Brief descriptions of the goal-directed activity of someone in an organizational role
	2. Use Cases	Detailed descriptions of interactive system usage to accomplish an organizational goal
	3. Scenarios	Detailed descriptions of a specific instance of system usage – may correspond to a path through a use case
	4. Test Specs	Detailed description of a system usage scenario with setup, wrapup, and checking steps
Behaviour Models	1. Decision Tables	Detailed descriptions of the system actions resulting from a complete set of logical conditions
	2. State Transition Tables	Detailed description of the system actions and entity states resulting from various logical conditions and trigger events including initial entity states
Facts	1. Constant Conditions	Detailed descriptions of conditions that must be preserved during all system activity
	2. Condition Dependencies	Detailed descriptions of the logical dependencies between conditions
Derivations	1. Derived Value	Detailed definition of a calculated value by specifying the calculation
	2. Derived Condition	Detailed definition of an abstract condition (for example potential in potential customer) defined by Boolean functions referencing the attributes of domain and system entities
	3. Derived Action	Work breakdown structure of an action
Definitions	1. Entity Specs a. Internal	Detailed definitions of domain and system entities that describe their structure, required and optional attributes, value ranges, units of value and relationships to other entities

	b. Input c. Output	
	2. Action Contracts	Detailed definitions of a system action using pre, post, and constant conditions
	3. Quality Specs	Detailed definitions of measures for assessing non-functional requirements
	4. Common Descriptions	Brief natural language descriptions of terms or phrases
	5. Acronyms	An acronym and the phrase it stands for

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.

Bleach & Soap 1: Magic Bullets 0

The American Journal of Tropical Medicine and Hygiene allegedly (QED, Robert Matthews, The Sunday Telegraph, 3 April 2005) recently reported a major success -- a 71% reduction in a major killer disease among children (2 million deaths per year) in a trial area, using two chemical preparations: sodium hypochlorite, and a mix of stearic, palmitic, and oleic acid. Or bleach and soap as experts prefer to call them; the killer disease was diarrhoea. Glamorous? Hardly. Effective? Certainly, but people seem to prefer magic bullets to scrubbing floors and washing hands.

Bleach and soap have the following characteristics:

1. unexciting
2. older than God (well, almost)
3. cheap
4. yes, they work
5. no short cuts, you get down and scrub
6. no special tools needed
7. and did I mention that they work?

What are the RE equivalents of soap and bleach, plain things that are known to work? The RE equivalent of magic modern drugs that will abolish infectious disease within 5 years is unfortunately all too easy to identify; it makes a fine pub game. But if you prefer practicality to fashion, here are ten simple things that certainly do the job.

Find out and document (as simply as you can manage):

1. the mission & scope
2. who the stakeholders are
3. their goals
4. how those goals conflict
5. scenarios to deliver those goals
6. requirements to get those scenarios
7. justification for each requirement (why it's needed)
8. assumptions underlying the requirements (existing systems, interfaces, competition, market, risks, etc)
9. agreed priorities on the goals/ scenarios/ requirements (to resolve the conflicts) -- this means review (I told you this wouldn't be exciting)
10. acceptance criteria for the requirements (how you know you've got what you asked for).

All of these can be written down with nothing more elaborate than a whiteboard, or a pen and paper. They can, if things get large and complicated, be written down using RM tools, or modelled in more elaborate ways. The point may be worth labouring, if only to emphasize that further research is NOT needed – the bleach works just fine, thankyou, as long as you scrub the bathroom with it regularly. So here is a table showing how the ten things might look on small and large tasks.

Work Product	Activity on Simple Project	Activity on Complex Project
1. Mission & Scope	write one statement, draw a context diagram	characterise the mission, draw a context diagram, agree it at the top (political) level across all decision-making stakeholders
2. Stakeholders	list them, maybe using a template	make an onion model; identify roles in each slot
3. Goals	list the goals, maybe make a hierarchy using Word outliner	make a goal model, indicate and/or decomposition, show support and obstacles (+/-)
4. Conflicts	list any conflicting goals, disagreements	classify conflicting viewpoints from the onion model, possible goal conflicts from the goal model by type and seriousness
5. Scenarios	write the essential operational scenarios as basic stories	create a set of scenarios at each level (business, system down to sub-sub-…system as appropriate); possibly structure them as use cases; consider negative and hostile scenarios, and counter them
6. Requirements	write a list of things the contractor /supplier/ programmers etc have to make the system do (and don't forget required qualities and constraints)	write and organize the requirements into a suitable structure, including functions, interfaces, qualities, and constraints; model the required behaviour to demonstrate its correctness and feasibility; trace back to goals and scenarios; allocate to suppliers / subcontractors etc

7. Justifications	write a few words justifying each requirement	make a cost/benefit model, simulations of performance, etc as needed to justify each requirement
8. Assumptions	list things that have to be true for the system to work	model the argumentation for and against the system, including but not limited to its safety case, showing assumptions, evidence
9. Priorities	rank the requirements into vital-desirable-nice to have (or 1-2-3, etc). Do the vital ones first	identify dependency classes (eg requirements involved in the same scenario); prioritise each dependency class; trade-off requirements against design options, etc
10. Acceptance Criteria	identify how you'll know when the system meets each requirement	as for the simple project, but also plan the test campaign based on the scenarios and the environmental conditions and non-functional requirements; trace the test cases to the requirements

There's nothing terribly new here, is there? But it isn't soft scented soap; it's more like the hard, old-fashioned blocks of 'Savon de Marseilles' that granny used to grate into the washing-bowl, than anything fancy from a chic boutique. Does your chosen gadget reach into all those germ-harbouring corners? It's interesting to take a patent RE method or two and see how many of the ten simple things they don't even consider. Try it.

It's also instructive to consider what happens when any of the ten simple things are overlooked.

1. No Mission & Scope? The scope steadily enlarges; the project zigzags from one possible purpose to another. Until it falls over.
2. No Stakeholders? Whole chunks of requirements are missed, only to be expensively inserted later; or else the project fails.
3. No Goals? The requirements are a disorderly unstructured ragbag, probably full of contradictions.
4. No Conflicts? They'll turn up later, at the most awkward moment, just as the project is running out of time and money.
5. No Scenarios? The system is probably unusable; it either never gets accepted, or it is shelved immediately on delivery. Even if the needed functionality is in there somewhere, it'll never work the way anybody wants it to.
6. No Requirements? Hrrm. You'll build the wrong thing. You'll have done that which you ought not to have done, and you'll have left undone that which you ought to have done, and there'll be no health in you, miserable offenders. Amen.
7. No Justifications? You'll probably build a whole lot of pointless features into the system because someone thought them up over coffee. You'll have made a whole lot of assumptions that seemed obvious to you but were invisible to your customer; and the users would at once have told you were wrong, if only you'd stated them.
8. No Assumptions? You and everybody else have certainly made some, and as the old business modelling joke has it, each one you make helps to make an ASS of U and ME. Assumptions are at their most dangerous when tacit, as everybody can then suppose that everybody else shares their assumptions (a particularly unlikely Utopia). In smarter language, any broken load-bearing assumption causes a failure (see the Book Review of Dewar's *Assumption-Based Planning* in *RE-Publications* in this issue of RQ, below).
9. No Priorities? Either the hard men will just chop your favourite requirements when the cash runs out, or they'll chop the whole project. You'll wish you'd make the real priorities clear from the start, and completed the vital bits before the axe fell.
10. No Acceptance Criteria? You'll have a whole lot of wishy-washy suggestions that won't be testable in place of requirements. You won't know when the project is complete, so contractual wrangling is likely. It'll end in tears.

There. Told you it would be old-fashioned. Or would you prefer some lilac-scented consultant-talk about endogenous quality-building and total responsibility-driven design, in fact the last thing in soft soap?

© Ian Alexander 2005

—oOo—

The following piece is a description of the capabilities of an 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.

Visualising the Requirements Process

Mark Walker, 3SL

At this year's INCOSE, 3SL is showing the latest version of our integrated Requirements Management and Systems Engineering product Cradle, with the emphasis on visualising the project process for faster time-to-start and minimal in-use overhead.

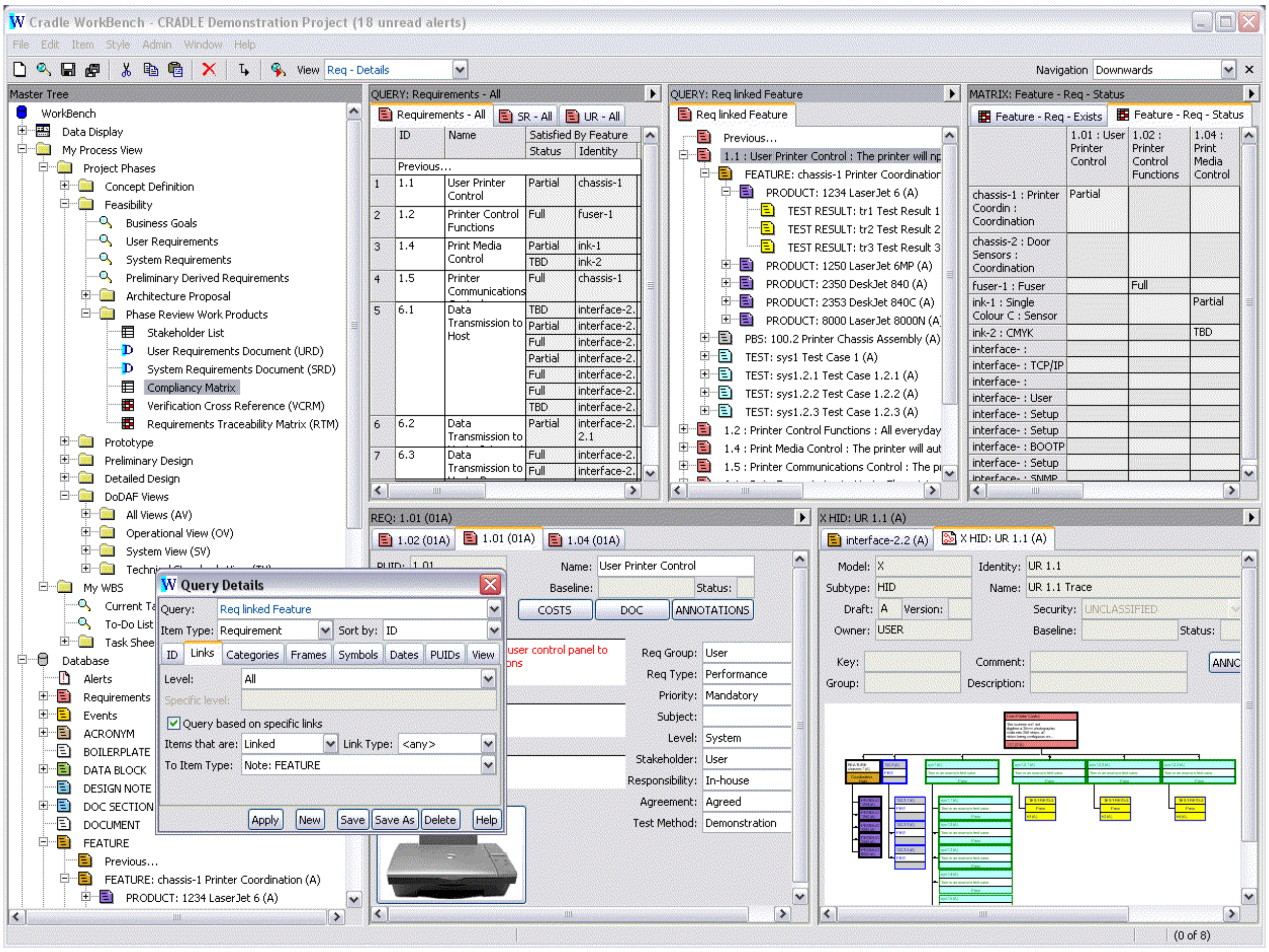
Cradle has supported both small and extremely large multi-site, multi-company projects for many years. Its uniquely extensible and truly multi-user database, point-and-click customization, massive scalability and excellent performance are the basis of its selection for some of the largest Requirements Engineering projects with a worldwide scope and 24-hour working.

Cradle databases have always provided user-definable object types, each with arbitrarily many attributes of any type. Attribute data can be held in Cradle, or referenced externally in files, at URLs, linked by Microsoft and other methods, or referenced in any external environment. Dependencies between items are reflected in links, optionally typed and grouped, and with searchable link attributes. Optional rules let the project apply the process consistently, down to individual users, user classes or groups if needed. Edit history controls track every change and update for all items, for all time, through and across Configuration Management baselines.

everyone understands how to follow the process with the tool, and respond flexibly as the project's and peoples' needs change. In doing so, the tool should intrude as little as possible. People are not interested in tools, they are only interested in getting the job done as efficiently and accurately as possible.

Cradle provides a user-definable process hierarchy as an integral part of its explorer views. This contains anything that the project, or users, want to see, in a structure that makes sense to them and uses their language and terminology. Perhaps you would show the project phases, with activities, phase reviews and the deliverables for each review. Alternatively you might show the project's work breakdown structure, with everyone's work shown in context. Or perhaps you'd show both, or something else entirely!

Each element of this hierarchy is linked to a query, a



Cradle WorkBench - CRADLE Demonstration Project (18 unread alerts)

View: Req - Details Navigation: Downwards

Master Tree
- WorkBench
  - Data Display
  - My Process View
    - Project Phases
      - Concept Definition
      - Feasibility
        - Business Goals
        - User Requirements
        - System Requirements
        - Preliminary Derived Requirements
        - Architecture Proposal
        - Phase Review Work Products
          - Stakeholder List
          - User Requirements Document (URD)
          - System Requirements Document (SRD)
          - Compliancy Matrix
          - Verification Cross Reference (VCRM)
          - Requirements Traceability Matrix (RTM)
      - Prototype
      - Preliminary Design
      - Detailed Design
      - DoDAF Views
        - All Views (AV)
        - Operational View (OV)
        - System View (SV)
  - My WBS
  - Database
    - Alerts
    - Requirements
    - Events
    - ACRONYM
    - BOILERPLATE
    - DATA BLOCK
    - DESIGN NOTE
    - DOC SECTION
    - DOCUMENT
    - FEATURE
      - Previous...
      - FEATURE: chassis-1 Printer Coordination (A)
        - PRODUCT: 1234 LaserJet 6 (A)

QUERY: Requirements - All (tabs: Requirements - All, SR - All, UR - All)

	ID	Name	Satisfied By Feature Status	Identity
	Previous...			
1	1.1	User Printer Control	Partial	chassis-1
2	1.2	Printer Control Functions	Full	fuser-1
3	1.4	Print Media Control	Partial	ink-1
			TBD	ink-2
4	1.5	Printer Communications	Full	chassis-1
5	6.1	Data Transmission to Host	TBD	interface-2.
			Partial	interface-2.
			Full	interface-2.
			Partial	interface-2.
			Full	interface-2.
			Full	interface-2.
			TBD	interface-2.
6	6.2	Data Transmission to	Partial	interface-2.2.1
7	6.3	Data Transmission to	Full	interface-2.
			Full	interface-2.

QUERY: Req linked Feature
- Previous...
- 1.1 : User Printer Control : The printer will np
  - FEATURE: chassis-1 Printer Coordination
    - PRODUCT: 1234 LaserJet 6 (A)
      - TEST RESULT: tr1 Test Result 1
      - TEST RESULT: tr2 Test Result 2
      - TEST RESULT: tr3 Test Result 3
    - PRODUCT: 1250 LaserJet 6MP (A)
    - PRODUCT: 2350 DeskJet 840 (A)
    - PRODUCT: 2353 DeskJet 840C (A)
    - PRODUCT: 8000 LaserJet 8000N (A)
  - PBS: 100.2 Printer Chassis Assembly (A)
  - TEST: sys1 Test Case 1 (A)
  - TEST: sys1.2.1 Test Case 1.2.1 (A)
  - TEST: sys1.2.2 Test Case 1.2.2 (A)
  - TEST: sys1.2.3 Test Case 1.2.3 (A)
- 1.2 : Printer Control Functions : All everyday
- 1.4 : Print Media Control : The printer will aut
- 1.5 : Printer Communications Control : The pr

MATRIX: Feature - Req - Status (tabs: Feature - Req - Exists, Feature - Req - Status)

	1.01 : User Printer Control	1.02 : Printer Control Functions	1.04 : Print Media Control
chassis-1 : Printer Coordin : Coordination	Partial		
chassis-2 : Door Sensors : Coordination			
fuser-1 : Fuser		Full	
ink-1 : Single Colour C : Sensor			Partial
ink-2 : CMYK			TBD
interface- :			
interface- : TCP/IP			
interface- :			
interface- : User			
interface- : Setup			
interface- : Setup			
interface- : BOOTP			
interface- : Setup			
interface- : SNMP			

REQ: 1.01 (01A) (tabs: 1.02 (01A), 1.01 (01A), 1.04 (01A))

PUID:	1.01
Name:	User Printer Control
Baseline:	
Status:	
Req Group:	User
Req Type:	Performance
Priority:	Mandatory
Subject:	
Level:	System
Stakeholder:	User
Responsibility:	In-house
Agreement:	Agreed
Test Method:	Demonstration

Buttons: COSTS, DOC, ANNOTATIONS

user control panel to ...ons

Query Details

Query:	Req linked Feature
Item Type:	Requirement
Sort by:	ID
Level:	All
Specific level:	
☑ Query based on specific links	
Items that are:	Linked
Link Type:	<any>
To Item Type:	Note: FEATURE

Tabs: ID, Links, Categories, Frames, Symbols, Dates, PUIDs, View
Buttons: Apply, New, Save, Save As, Delete, Help

X HID: UR 1.1 (tabs: interface-2.2 (A), X HID: UR 1.1 (A))

Model:	X	Identity:	UR 1.1
Subtype:	HID	Name:	UR 1.1 Trace
Draft:	A	Version:	
Security:	UNCLASSIFIED		
Owner:	USER	Baseline:	
Status:			
Key:		Comment:	
Group:		Description:	

(0 of 8)

Cradle provides list, tree, table, hierarchy and matrix views that can present the relationships between any number of item types, all specified by a point-and-click user interface. Combined with user-defined reports and documents, this provides a highly flexible information presentation engine, all user-definable without any need to resort to a programming manual.

But whether large or small, many of the issues that confront projects using tools are the same. They need to define a process, map it into the tool, ensure that

matrix, a report, a document, or simply a part of the database; whatever you need. The point is simple, you see something that you understand and can use without manuals or training. Just browse the hierarchy and the data you need is shown in the manner you need, on screen, as a generated document, as paper coming out of the printer, whatever is most efficient.

And better still, you can link Cradle to external tools through a range of interfacing technologies best left for your IT department to worry about, or through an

Application Programming Interface if you really want to get down into the detail.

The same facilities and concepts apply to web and non-web clients. Cradle supports any number of user-defined web user interfaces, each presenting an interface with the features and complexity appropriate to the needs of each user group to help you to get the information to everyone, however occasional or specialised their use of the data may be.

Remote working is increasingly common, and Cradle fully supports it with distributed databases and a full suite of networking options including HTTPS, VPN, SSL and the rest of the alphabet soup of acronyms.

And the support not does end with the requirements, but continues through the lifecycle with totally integrated analysis, architecture and design models, performance assessment, risk assessment, built-in Configuration Management and Change Control, and the project administration and infrastructure support that you expect and need.

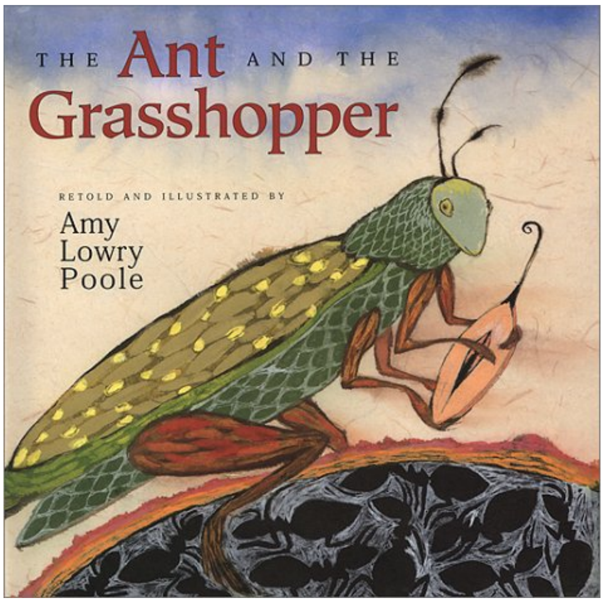
3SL is totally committed to open standards, reflected in Cradle's support for frameworks and standards such as DoDAF, AP233 and XMI, a myriad of data converters and translators, and a technology base of both Windows and UNIX (including Linux) for all of the Cradle clients and servers.

Cradle is designed and built in the United Kingdom, driven by our customers' needs, and supported worldwide. It delivers a truly integrated solution that spans your process, integrates with your environment, and empowers you to get the job done in the way that you want to do it, quickly, easily and simply.

© Mark Walker (3SL) 2005

RE-flections

The View from Enemy Territory, or, The Ant and the Grasshopper



The cover of '*The Ant and the Grasshopper*'
(a fable of Aesop, 6th Century BC),
retold by Amy Lowry Poole, Holiday House, 2000

RESG members are familiar with the general argument for RE, and for Systems Engineering in general.

It runs (as you know) that over half of all current projects fail to deliver on time, to the right quality, to budget. Project managers, when asked, report that over half the reasons for this concern requirements. Therefore, the argument runs, project managers must ensure that much more money, time and effort are spent on requirements. This will save them time, money, and embarrassment later.

Put another way, to err is human. Everybody injects mistakes into their work. Programmers do it; designers do it; analysts do it; even users trying to state requirements do it. So, there is a fixed number of bugs – let's call it B – in a given size of system. Project managers can choose to do the bug fixing late, say at testing time, or early, say at specification time. Why should anybody care which it is? Is it actually better to be an eager ant, working through the summer, rather than a lazy grasshopper, singing while the sun shines? Why bother to do a whole lot of tiresome requirements elicitation and analysis, in fact?

Well, the marching army is large, late on in the project, so more warm bodies are drawing salaries, chewing pencils, writing on flipcharts and generally trampling the fields of Flanders into mud than early in the project. So, the cost of each bug's worth of delay is high. And, each bug is embedded in a bit of system which is connected to some other bits of system, which are documented in a lot of bits of specification and test plan, and … you get the picture. So there is a whole heap of rework for the grasshoppers to do. What's more, every day of delay causes a bit more damage to market share, customer impatience and the bottom line. The overall cost of each bug – let's call it C – is thus high in more ways than one. So, the total price of the grasshopper approach is the painful B x C, where B is constant and C is very large.

On the other hand, the ants who eagerly discover and fix all the bugs (if you take my meaning) as early as possible do so while the project is small, there's almost nothing to rework, and the world's press is not camped on their doorstep. Therefore the total price of the "left-shifting" ant approach is the painless B x c, where c is the very small first-time cost.

The ant approach is therefore much cheaper, faster, and better than the grasshopper approach, QED (the chief ant takes an elegant bow at this point), and how many licences for my patent Requirements Management tool did you say you wanted?

Whoa there! What about the other side of the argument? How does it look from the project manager's point of view?

Rather different. Here is a marching army of ants, carrying all sorts of picks, shovels, and heavy earth-moving equipment that we didn't know we needed. The chief ants recommend immediate expenditure of 15% of the total project budget, several months of hard work, and at the end of that they won't have written a line of code!

What's more, they want to spend that money NOW. I'm probably going to get promoted/ moved sideways into Corporate Configuration Management/ pregnant/ emigrate to Australia/ etc. The project is very likely going to get reshaped/ cancelled/ reorganized/ etc when the company gets taken over/ merged/ outsources its software development division/ etc. So who knows if anyone will ever get around to finding the bugs these JCB-wielding ants say must be in there. My money is here now – and the future, well, it may happen or it may not[1]. Anyway, by then we'll probably have a bit more spare cash to spend on fire-fighting, whoops I mean bug-fixing.

Furthermore, who knows if the ants are right about all those bugs anyway? My team are following Industry Best Practice and we have a fine reputation. So we're with the grasshoppers. What d'you mean I'm planning for failure? Where's that violin now?

© Ian Alexander 2005

1. Economists call this 'discounting the future' and they do it rather like calculating the annual interest on a loan. If testing is 1 year away and we discount at 10% p.a., the cost of bug-fixing is only (B x C) x 0.9. If it's 3 years away, the factor is 0.9^3 (= 0.729). The further away something is, the smaller it looks, in fact.

Special Guest Proverb

This issue's proverb clearly has to stray into enemy territory, but which side of no man's land it may be (ant or grasshopper country) is a matter of opinion.

We'll cross that bridge ... when it falls down.

The Team Leader's Maxim

RE-Creations

To contribute to RQ please send contributions to Ian Alexander (ian @ scenarioplus.org .uk).
Submissions must be in electronic form, preferably in Word 2000. Deadline for next issue: 15th September 2005

RE-Publications

Assumption-Based Planning

By James Dewar

Cambridge University Press, 2002

ABP is a technique used by business executives to peer into the future. In so doing, it deals with assumptions, things that are taken (rightly or wrongly) to be true in the world, and actions, things that are to be done if the assumptions come true, and equally or more importantly, things that are to be done if they do not. There is an obvious parallel (to a requirements person anyway -- perhaps the first law of assumptions is that obviousness varies!) with requirements and domain statements in the world, and specifications for designing and building a system.

ABP comes from the RAND corporation, home of the use of imagined 'alternative future worlds' or static Scenarios for planning. It is like and yet unlike; it focuses not on the future worlds themselves, but on what assumptions underlie your imagination of the future.

Dewar claims to be an unwilling author, preferring to be a practical mathematician and consultant; but he writes clearly, fluently, and persuasively. The key concepts are elegantly few, and they serve as powerful conceptual tools for improving plans. ABP, in fact, is not so much a planning process as a re-planning process: it sharpens plans by questioning them and forcing planners (and more importantly, decision-makers) to look at how their plans could fail. That is in terms of plausible events that could break the assumptions on which the plan rests.

There are two components to risk -- the probability of an event (such as a failure), multiplied by its seriousness should it occur. Remarkably, Dewar lays aside all his mathematics and simply focusses on identifying which assumptions matter, ie are 'load-bearing' for the plan; and which could plausibly fail, ie are 'vulnerable'. These are the key assumptions. There is one assumption underlying this approach: that failures are single-point, so that the failures of assumptions can be considered one at a time. This won't do, of course, for complex safety-critical systems, where all the easy paths to failure were taken care of long ago, and five or more causes are now needed to create a serious accident. But Dewar knows what he's doing when he says that you can't calculate probabilities for single events.

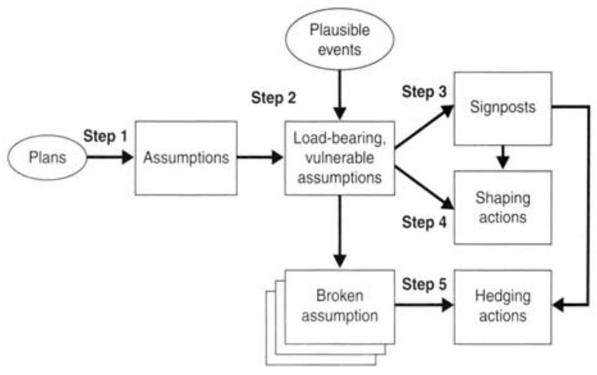


Figure 1.1. The Basic Steps and Flow of Assumption-Based Planning

Having identified plausible events, the planner's task is to identify suitable Signposts that warn that these events are taking place, so that something can be done about them. That something could be a shaping action, if the organisation is capable of shaping events in some way; or a hedging action, to soften the effects if the worst should happen.

The book is narrated in an enjoyably reflective way, and illustrated throughout by the possibly rather too trivial example of a lemonade stand (which won't sell much if it's rainy on the day). Other illustrations, told anecdotally, are more serious, being from the Cold War and concerning nuclear planning for the defence and/or destruction of the world. On a happier note, Shell correctly identified its assumption that the oil price would not fall dramatically, and weathered the low oil price shock of the late 1970s much better than other oil companies through careful hedging actions taken in advance.

This is an excellent and stimulating book of obvious importance. A client said that it was better than many days of management consultancy; and to a requirements person, it is suggestive of a range of worthwhile and practical approaches for surfacing, modelling and documenting the reasons for building systems. Students, managers, planners, and systems engineers should all read Dewar attentively.

© Ian Alexander 2005

***RE*-Sponses**

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

RQ35 quoted the proverb:

> "Those who cannot remember the past are condemned to repeat it"

I must admit my favourite version is:

> "History repeats itself, it has to, nobody listens."

I think that one is thanks to a poem by Steve Turner.

The customised RE version was:

> "Those who have never heard of a good system development practice are condemned to reinvent it"

I wish that were true! Too often it's the bad ones that get reinvented...

Robin Evans

(robin.evans@unumprovident.co.uk)

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

subscribe RE-online <your email address>

LINKAlert:

http://link.springer.de/alert

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

RE-Actors: the committee of the RESG

Patron:

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

Chair:

Dr Pete Sawyer, Computing Department,
Lancaster University,
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 Lucia Rapanotti, Computing Department, The Open University,
l.rapanotti @ 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 (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

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 member (£10)	[ ]	*BCS/IEE members, please 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: Dr Lucia Rapanotti
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.