Requirenautics Quarterly

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

http://www.resg.org.uk

©2003, BCS RESG Issue 28 (February 2003)

RE-Locations

RE-Soundings

Editorial

Welcome to the first RQ of 2003. With the 11th IEEE International Requirements Engineering Conference (RE'03) coming up in September, daily illuminating debate on the SRE mailing list and a continuing stream of new RE books, the signs are that the struggle to win recognition of the importance of RE is still being won. Or at least they are from my perspective in Lancaster University. Perhaps it looks different to beleaguered practitioners in these icy corporate times. It would be good to get a practitioner's view of what arguments have been won and what current pressure-points are RE-wise in industry at present. All contributions on this and any other relevant subject are most welcome.

One of the interesting recent developments in Computing is the emergence of AOP. It's too early to tell if this represents a significant new programming paradigm but it's certainly interesting. In the same way that object-oriented ideas migrated to (or were co-opted by) RE, people are now investigating whether aspects represent a way of thinking about and modeling problems, business processes and systems. We've known that many requirements and constraints are widely scoped and that these often pose particular problems, for a long time. There's an obvious parallel here with cross-cutting concerns that form the focus of aspect-oriented approaches. The interesting question is whether what the AOP community has learned form supporting these directly at the implementation level holds anything that can be usefully applied at the RE level.

You'll find a call for papers for a special issue of *IEE Proceedings - Software* later in this issue on this very subject: *early aspects*. Even if you aren't submitting, it could be an interesting trend to look out for in the next few years.

Pete Sawyer
Computing Department, Lancaster University.

Chairman's message

Greetings! We're well beyond January for me to use this message to wish you all a very Happy New Year, so instead I'd like to thank you all for renewing your membership for 2003. You will not have missed the fact that we have resumed charging nominal membership subscription fees this year, after providing free membership as a one off in 2002, in order to re-organise our membership database. Thanks to Steve Armstrong and his membership team for getting this done!

I hope that you will continue to find membership of the RESG worthwhile; your RESG Executive Committee is working hard to put together a varied programme of events for 2003/2004, which we hope will encourage you to attend (or, at the very least, to read all about them in your copy of the RQ newsletter).

This is an ideal time for you to send us your ideas and suggestions for event topics, formats, and venues, so please do let us know what you need, and we'll try to implement a programme of activities that meets your real requirements, rather than leaving us to guess what those requirements might be.

I'd like to end on a personal note. Congratulations to RESG committee member Alessandra Russo on the birth of her second baby girl, Natascia, and welcome back to the committee to Efi Raili who has just returned from maternity leave!

Bashar Nuseibeh
The Open University

RE-Treats

For further details of all events, see www.resg.org.uk
Next event organised by the group.

COTS Integration: Why You Need Requirements

Date: 9th April 2003 (afternoon)
Location: Room F211/F212, Feeney Building, Perry Barr Campus, University of Central England, Birmingham
Contact: Ian Alexander (iany@easynet.co.uk)

This innovative event format is in 2 parts. In the first part David Bush of National Air Traffic Services Ltd, will introduce a safety-related COTS (Commercial-Off-The-Shelf) challenge for the systems and requirements engineering process. In the second part three leading academic research groups will show how their processes, techniques and tools could address the particular problems and issues the challenge presents. The speakers and their affiliations are as follows:

Professor Neil Maiden - City University London
Professor Ian Sommerville - Lancaster University
Ms Carina Alves - University College London
Dr Ljerka Beus-Dukic - University of Westminster

Titles and abstracts of the individual presentations to be announced soon.

The event will end with a wide-ranging discussion on requirements engineering and COTS software. It will be relevant to anyone interested in how to select, customise and integrate complex off-the-shelf solutions in software and systems engineering and, in particular, in the role of requirements in such integration. The event will be also of relevance to those involved in procuring complex systems, e-business solutions and software components.

Other events likely to be of interest to RESG members.

Extending Requirements: A Practical Workshop

Date: 28th-29th April 2003
Location: London
Contact: Jeanette Hall (Jeanette@irmuk.co.uk)

Taking your requirements to the next level. This workshop shows you how to build on your existing expertise and integrate requirements into project management for the maximum effect. RESG members are entitled to a 10% discount.

www.irmuk.co.uk/58.htm

Mastering the Requirements Process

Date: 6th-8th October 2003
Location: London
Contact: Jeanette Hall (Jeanette@irmuk.co.uk)

A 3-day seminar and workshop. RESG members are entitled to a 10 percent discount.

www.irmuk.co.uk/1.htm

RE-Calls

Recent Calls for Papers and Participation

11th IEEE International Requirements Engineering Conference (RE'03)

8th - 12th September 2003, Monterey Bay, California USA

http://www.re03.org

The RE conferences are a platform for research to present novel results, for transfer of research results to industrial practice, and for the presentation of industrial experiences that can inform new research directions. Two kinds of technical papers can be submitted: research and experience. Topics of interest include, but are not restricted to:

- Requirements elicitation techniques
- Requirements validation techniques
- Requirements management and traceability
- Requirements evolution
- Requirements, software architecture and business architecture
- Requirements prioritizing and negotiation
- Combination of formal and informal specification techniques
- Requirements for high-assurance systems
- Making formal techniques usable
- RE for mechatronics systems
- Specification of quality attributes
- Requirements metrics
- Tool support for RE
- Prototyping, animation and execution of requirements
- Requirements for business systems (workflow, groupware, e-commerce systems)
- Requirements for web-based systems
- Requirements for ubiquitous computing
- Requirements for product families
- Requirements engineering case studies and experiences
- Cognitive, social and cultural factors in RE
- Requirements engineering education

The submission date for papers and industry track contribution has now passed. See below for deadlines for workshop proposals, tutorial proposals, doctoral symposium papers, posters and research demos. For any other queries, please contact info@re03.org.

Key Dates

- Paper submissions (passed)
- Poster submissions 26th April 2003
- Workshop proposal submissions 29th March 2003
- Tutorial proposal submissions 29th March 2003
- Doctoral symposium submissions 29th March 2003
- Research demo submissions 26th April 2003
- Industry track submissions 1st March 2003

Ninth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ ’03) In connection with : CAiSE’ 0 3.

16 - 17 June, 2003 Klagenfurt/Velden, Austria.

http://crinfo.univ-paris1.fr/REFSQ/03/

At eight previous REFSQ workshops, researchers and practitioners from various disciplines have contributed to improve the definition and implementation of quality requirements. REFSQ’03 will continue this tradition as a highly interactive stage for discussion of quality-related problems in RE as they have developed over the last years. In particular, we encourage people from the requirements engineering, software engineering and information systems fields to present their approaches to higher software quality and to discuss how requirements engineering can contribute to it.

REFSQ’03 invites general submissions addressing a wide range of RE issues, such as:

- understanding and improving RE-processes;
- new methods and method engineering for RE;
- empirical evaluation of RE methods and tools;
- empirical studies of industrial RE practice;
- transdisciplinary theories of and paradigms for RE.

In addition, the REFSQ’03 special theme is : *Integration of Requirements Engineering into Software Engineering.*

Requirements are used for various purposes in software engineering and project management. We seek, as is the tradition in REFSQ, for original RE methods, techniques and tools aiming at improved software quality. Possible topics include, without being restricted to:

- integration of RE with project management activities (e.g., project planning & controlling)
- integration of RE with quality assurance activities (e.g., inspection, testing)
- relation of requirements and architecture,
- relation of RE and change management,

Researchers and practitioners are encouraged to submit papers complying with this special theme. Submissions dealing with the traditional REFSQ topics are of course welcome.

IEE Proceedings - Software. Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design

Guest Editors: Dr. Awais Rashid (Lancaster University, UK), Dr. Ana Moreira (Universidade Nova de Lisboa, Portugal), Dr. Bedir Tekinerdogan (Bilkent University, Turkey)

Aspect-oriented software development (AOSD) techniques aim at providing means for the systematic identification, modularisation and composition of crosscutting concerns throughout the software life cycle. The aim of this special issue is to facilitate the cross-fertilisation of ideas in requirements engineering, software architecture design and aspect-oriented software development.

Crosscutting concerns also exist at requirements and architecture level mainly in the form of broadly scoped properties and constraints. However, research on the use of aspects at these early development stages has received far less attention to date compared to work on detailed design and programming approaches. The term

"Early Aspects" denotes a requirements engineering and architecture design approach that integrates aspect-orientation techniques. From a requirements engineering and architecture design perspective such cross-fertilisation will improve and broaden the understanding of the identification and management of requirements and architecture level concerns. This will support the management of concerns at the requirements analysis and architecture design level that tend to be more systemic, cannot be localized in single components and as such crosscut the functional architecture and requirements components. From an aspect-orientation perspective this will lead to a better understanding of how aspects can be used to support systematic and rigorous development of software from the very early stages such as requirements engineering and architecture design.

The deadline for submission of manuscripts is 02 June 2003. For a detailed list of topics of interest for the special issue and submission guidelines please refer to: http://www.iee.org/Publish/Journals/Profjourn/Proc/sen. Alternatively, contact Dr. Awais Rashid (awais@comp.lancs.ac.uk).

RE-Readings

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

Using Formal Models to Understand Requirements Better

RESG Event, November 6 2002, Imperial College London.

We had two chairpersons for this event (or three if you count Alessandra's 6-weeks-from-term baby), **Bashar Nuseibeh** and **Alessandra Russo**. They were delighted to welcome an all-tickets-sold audience to Imperial College and the RESG.

Axel van Lamsweerde gave a morning tutorial on Goal-Oriented Requirements Engineering. He was on top form and gave an enjoyable and readily-understandable overview of RE from the point of view of goal modelling. Goals are the why; requirements and assumptions are the what, he said. Goals had been ignored by UML, but guru Fowler agreed they were needed. (Mind you, guru Cockburn treats use case titles as a hierarchy of functional goals.) Goals were prescriptive (optative in Jackson's parlance), unlike domain properties which were descriptive (indicative). In other words they were essentially high-level requirements. They didn't have to be functional; non-functional goals could be for safety, security and so on: what the Toronto i* people call softgoals. He argued that there wasn't much of a distinction between functional and non-functional goals/requirements, and it is true that there is an interplay between them. For instance, a security NFR causes various functions to come into being, e.g. to lock things and to record accesses. But that doesn't make the NFR the same as the functions that implement it, does it?

Agents always went with goals; whether human or machine, agents co-operated to satisfy goals. Perhaps more contentiously, goals had to be realizable, at least at finer levels of detail when you get down to defining exact logical conditions for them – low-level goals were thus implicitly equated with requirements; high-level goals were what some people call objectives.

Goal modelling could proceed top-down (the obvious approach) or any other way; ANDing subgoals explains the basic rationale for high-level goals; subsequent OR-abstraction allowed you to refine the alternatives. This analysis helped to discover conflicts, early in the life-cycle – long before design – by being declarative about what people wanted.

On the more formal side, you could show that a set of requirements was "complete" if for all goals, the requirements, assumptions and domain knowledge entailed the goals. Obviously this could only work if the goals were realizable 'user requirements'. Similarly, a requirement was only pertinent if it helped to satisfy a goal. But you absolutely couldn't show that a set of goals was complete with respect to the real world. Low-level goals could be verified; high-level objectives could in Herbert Simon's unlovely phrase be 'satisficed' to some degree of confidence.

Goals at a high level were generally more stable, as Annie Anton had observed, so in principle you might be able to use goals to document product evolution (e.g. in product lines). In theory you'd do this by providing new alternative ways to satisfy a stable goal. I think this may be a tall claim because old product features can always become obsolete; cars used to have to be able to tune to Medium Wave and to play Audio Cassette tapes, for instance; while it may be true that high-enough objectives ('play music') are indeed stable, they aren't very useful as product requirements.

Obstacles allowed you to reason about 'unexpected agent behaviour' in a goal model. Formally, an obstacle obstructs a goal if and only if the presence of an obstacle in a domain meant that the goal could not be achieved; and that the domain definition did not exclude the possibility of the obstacle's existence. Identifying and removing obstacles helps to make requirements more complete, more realistic, and hence more robust. However, this happy story does not take into account the possibility that malicious agents can be creatively hostile: there is an arms race between your obstacle-removing tactics and the subsequent moves of the other side. 'Obstacle' sounds and is static; a more dynamic approach is to deal with threat and counter-threat as a game which won't necessarily end, and in which new moves are expected.

Thoughts of obstruction led naturally to consideration of safety and security issues, including the famous counter-example of aircraft wheels aquaplaning on the runway to the simple assumption that the plane is on the ground and may use its reverse thrust if and only if its wheels are turning. Jawed Siddiqi (Sheffield Hallam) said this was all post-hoc, and asked how you would probe to elicit such things? Axel replied that indeed there was "no free lunch" – you get completeness only with respect to what you already know about a domain. "We use obstacles all the time, semi-formally – using patterns to provide proof", said Axel. (Anthony Hall said over a cup of tea that safety engineers are like old generals who always fight the last war; safety people always prevent the crash that happened last time.) Could goal-obstacle analysis then ever be predictive? Axel replied that the disastrous Uberlingen midair collision (July 2002) had in fact already been analysed in a large (>300 goal) model which had been completed back in March 2002. All the six or so "perversely chosen" obstacles that occurred simultaneously – there was only one controller, he was overloaded, the onboard collision warning system and the air traffic controller gave conflicting orders, etc – were all in the model but were in different goal trees. To be predictive, you'd have to be able to assign probabilities to complex combinations of obstacles.

Scenarios and goals offered complementary benefits; scenarios for elicitation and validation with people, goals for reasoning. Scenarios were excellently concrete and narrative, but were partial (like test coverage), procedural, and vulnerable to combinatorial explosion. But, said Axel, "we finish where the formal modelling guys usually start" – goal models formed a natural bridge between domains and systems.

In the afternoon we had four speakers and then a panel session.

Jeff Kramer (Imperial College) talked about Scenarios to Behaviour Models, and Back. He felt that behaviour modelling should be part of everyone's requirements process: you needed to go from requirements to models to make things definite. Scenarios were wonderful for elicitation and for explaining why bits of behaviour were needed, but they had no precise semantics, even when they were as neatly documented as a UML message sequence chart. A model was complementary to a set of scenarios; it could be analysed and checked; it could be animated to generate a trace (i.e. a scenario) or drawn as some picture of the system which you could then animate in a more visual way as a sort of simulation.

Equally, you could go in the other direction. "People find message sequence charts easier to produce than models", he said. So, how could you create models from scenarios? If you treat scenarios as sequences of activities, you can eaily put together a state machine. If scenario 1 goes ABCD and scenario 2 goes ABCE then state C has two exits, and so on. In general the resulting state machine or architecture is richer than the traces or scenarios as you start discovering implied scenarios. If scenario 3 goes EDFG then you discover that ABCEDFG is possible, as well as ABCDFG which you didn't know before. You may discover ways of getting into error states, or things you don't understand that you can check with your stakeholders – using animation of scenarios to give them a picture of what the model implies.

Like Axel, Jeff said there was "no free lunch" – the approach did not discover unknown event types. What it could do was discover negative scenarios, certain implied scenarios that shouldn't happen.

Axel van Lamsweerde asked about the quality of models if they were based on instances. Jeff Kramer replied that when you went to negative scenarios (after the initial model-building), you often needed to generalize: "that's an instance of the general case, one of a class of counterexamples".

Anthony Hall asked if you put multiple scenarios into one behaviour model? Jeff Kramer replied that the approach did exactly that. It could handle 10,000 states which was rich enough for significant problems. Finite state modelling was vulnerable to combinatorial explosion of scenarios but it was tractable.

Paolo Giorgini (Trento, Italy) talked about Reasoning with Goal Models in the Tropos Methodology. AND/OR trees (a la Axel) could not handle vague or partly-defined goals. You could have networks of goals with cycles of dependency (positive or negative), for instance. Tropos reasons about full or partial evidence that a goal is satisfied or denied. Intuitively these can be based on positive and negative relationships between goals held by actors. The reasoning is fully axiomatized and can be applied qualitatively or quantitatively. Tropos can propagate values (full, partial, none) or -1.0 .. + 1.0 across a 30-goal network in negligible time (1/100 second). You can then see which relationships are important.

Tropos - movement - first modelled the environment as Actors with dependencies on each other. Then it took the 'system actor' and modelled its dependencies – whether these represented functional or non-functional requirements – on actors in its environment. Then it decomposed the global system into subsystems and modelled the data, control (etc) dependencies between these. Finally, it analysed the details of input, output, and control. In this way it aimed to bridge the gap between the early-in-the-life-cycle approach of i* and the late-in-development approach of agent-oriented programming. KAOS was another early approach; UML mainly later. Tropos was unique in using the notion of agent throughout development.

Anthony Hall (Praxis) talked about Industrial Experience with Formal Requirements. Formality added precision and enabled exact reasoning. Did it really help, he asked? "It really makes the whole development more visible" he said. He explained the basis of the Reveal method, and gave some examples. Requirements Engineering was about showing that the Requirements were entailed by the Domain and the

Specification, i.e. that the requirements would be satisfied. (In response to a question, he agreed that his requirements were Axel's goals, and his specifications were Axels requirements.) The domain could be modelled formally by analysis, with business rules, laws of physics and so on. By being precise, early, you could discover problems and reduce trouble during development. For instance, smart card issuers were naughty in assigning card IDs so you in fact needed to know the issuer, the ID, and some sort of hash function to identify the application correctly.

Similarly in requirements you often (but not always) needed to be formal, especially when safety or security were concerned. On the other hand, vague high-level objectives like improve throughput might well be acceptable. Precise requirements could be defined in Z, a logic notation. You could then reason about satisfaction and detect errors and missing requirements. Contrary to expectation, this was cheap; Z found as many errors as unit testing, but was five times cheaper. It found errors early too: many of the errors introduced by specification were removed during architecture or detailed design, and in a recent project only one specification error survived through to operations. Modestly but with some pride he quoted an authentic customer statement: "The system behaves impeccably as expected."

Axel van Lamsweerde said that he agreed 1000%, but what of people who advocated agile methods? Anthony Hall replied by quoting John Barnes: "Ada is not meant to make programming quick. It's meant to make programming slow. Slow is good". There was laughter.

Tim Clement (Adelard) talked about Dust-Expert™: an example of formal methods use in industry. Almost any powder like sawdust, flour or custard could explode if mixed with air and ignited by a spark – electrostatics were likely in a dust-filled factory. He showed photographs of a furniture factory with all its walls blown out. He had used VDM, another notation like Z, to define the specification of Dust-Expert precisely. Numerous useful properties of the system emerged from the formalism; for instance, the Graphical User Interface revealed all of the system's internal state, and essentially always (ok, not while the user was updating a field) exactly represented it. So you got properties like interface predictability and correctness of representation. Productivity was above average for safety-related software,and there were only 1.5 errors per 1000 lines of code (a total of 42 bugs). 31of these were discovered through testing, and the rest through field experience.

The **panel session** was chaired by Bashar Nuseibeh. He asked how to justify formal models in industry. Tim Clement said it was cheaper on maintenance. Anthony Hall said that if the customer cared if [a system] worked, it was the fastest and cheapest way, e.g. for embedded software. He agreed that if you didn't care, you didn't do it – a bank made a cold-blooded calculation that getting a website up 2 months early got you 200,000 extra customers, and it then cost you only £20M to fix the problems which was worth it. "We're a bank, we always do this."

Specification of Distributed Systems Security Policies Dr Emil Lupu (Imperial College)

Part of the IEE Summer School on Distributed Systems

September 5th 2002.

Issues are still at the border of research and product development: from role-based access control, trust, policy-based authorisation, and conflicts/exceptions. At Imperial the focus is quite practical.

By trust, some people just mean reliability or dependability; others something quite different.

Access control needs to work in very large complex networks, with many machines hosting services. There are millions of objects, tens of thousands of users and many platforms and services. Permissions must therefore be very fine-grained compared to the network – who can invoke what service on what platform; able to change rapidly; and able to cope with heterogeneity. That's bad enough: but what about laptops, web-enabled cellphones, handheld devices and so on? Visitor policies are needed to ensure security.

The solution is in having defined groups and roles, middleware in CORBA or Java that is now much improved; security monitors, and a policy that spells out what to do when something that looks like an intrusion happens.

Role-Based Access Control (RBAC) means tying permissions to roles instead of individual users. You define roles – like groups, and you can then dynamically assign users to roles without having to redefine those roles each time. The indirection (user – role – permission) affords more flexibility. . Many tools such as Win 2000, Oracle, Tivoli, Sybase all claim to be role-based.

You can also have permissions from junior roles inherited by senior roles through tools that support role hierarchies – but this is a more dubious advantage, argued Lupu: was it true that professors should inherit permissions from lecturers who inherit from research students who inherit the permissions of a generalized department member? It might be right or wrong. Should the prof be able to read all the private files of all research students? The crude rule – senior roles aggregate all access rights of junior roles – needs to be modified by defining private roles (or projects). Hierarchy reduces the number of permissions as claimed, but can increase the number of roles needed. It also raises questions of complexity and efficiency if you have to go around a network to find roles to check permissions for a specific user. The model doesn't provide a language for specifying policy, and it is basically ad-hoc – every tool has a different approach.

Trust policies started with IBMs Trust Policy Language (TPL) which was a way of using public-key certificates over the internet to control access to resources by user groups; access rights are assigned to groups. TPL is based on XML syntax. Different authorities issue certificates and authorise accesses. Expired certificates have to be revoked. Certificates are valid for a year or so, creating a sizeable overhead of administrative complexity: the key problem in security. The certificate has to name the issuer, subject, type, version, and links to where the certificate syntax and other details are defined. TPL itself allows both positive and negative rules (but you have to be careful with the latter). E.g. you can define that customers need an employee of rank greater than 3 to sign their certificate. Then you can have access rules saying that customers can browse the catalogue and order goods, and so on. But the XML rules are verbose (the customer rule is a page of code!) and unreadable, and no concept of inheritance.

The Oasis consortium is trying to standardize on a different use of XML, with a language for specifying authorization policies (XACML), and a second language (SAML) for marking up security assertions (like "Joe Bloggs has been denied access"). This complies with the idea of separating policy decision-making from policy enforcement. The enforcement point may detect a possible security breach, and inform the decision point which can apply more resources to evaluate the situation. In a blizzard of three-letter acronyms starting with P, policies are handled by being passed around from policy administration to retrieval to decision to enforcement to information and back. Obviously the intended target is the web services market.

There's a major problem with usability; the stuff is just as verbose as IBM's XML, and there are no graphical or high-level tools yet before diving into the XML. Pity the poor ex-sergeant-major security officer who has to specify and enforce security! You can clearly create your own higher-level language specification in a database tool and write your own XML exporter; but only the XML format is currently defined – clearing up the current ad-hoc muddle. But Tool support and friendlier notation are clearly needed.

Policy-based authorisation has evolved over a decade. You basically organize a hierarchy of objects (such as computers) into a Domain, and apply a common policy to them all. Again, you can change domain membership without editing the policy – it's just like moving users into or out of a role group. Hierarchy enables you to scale up the application of a policy.

There are 2 basic types of policy: authorisation policies (what you can and can't do) and obligation policies (what you have to do when certain events occur). Negative authorisation is useful especially for revocation of access rights, and can also directly represent organisational prohibitions which are often far more common than permissions. Obligations are typically interpreted by the machine subjects themselves – e.g. when 3 login failures happen, you should disable the affected userID and notify the administrator. Other more specialised types of policy include 'filtering' (weed out confidential parameters) and 'refrain' (hold back from supplying information).

Conflicts can arise when 2 or more policies apply, e.g. when an object is in multiple domains; and because different managers can specify policies, e.g. security and performance policies for the same network. You obviously need a way of detecting conflicts and handling exceptions economically; you don't want to edit a policy every time you find an exception (e.g. students can't reboot servers; but student Smith may do so). A default is that negative rules always override positive ones (so hard luck on Smith). Or you can specify Priorities – but it's hard to stay consistent. A general concept is to evaluate a "distance" between a policy and the object it applies to; near overrides far. Or you can try more recent overrides older. Or more specific (exception) overrides more general (basic rule) – but this doesn't always work. But all of these (modality) conflicts are alas by far the easier type. The hard cases are semantic conflicts. For instance, self-management: a manager may not authorise his own expenses; separation of duties: one person may not initiate and authorize payments. Such meta-policies are very troublesome. You can try to describe them in languages like Prolog or OCL. There is a clear argument for having policies expressed in a declarative language – if all your security policies are encoded in Visual Basic, you can't do much to look for conflicts, gaps, or errors.

Future directions include refinement of trust-based policies; dynamic adaptation; mobile architectures (and pervasive computing, with wearable and invisible devices); and the reverse engineering of existing implementations.

Lupu was cogent and lucid throughout. The group (a dozen) seemed to have been happy with the other sessions they'd attended. There were some jokes about speakers bringing their usual 200 slides for a single session; every aspect of distributed systems seems to be complicated. The atmosphere was studious; the audience mostly male and casually-dressed. Everyone listened closely rather than looking at the neatly-printed sets of slides. There were one or two technical questions at the end of each topic; questions were answered cogently and honestly, pointing up the key issues. It seemed that the Summer School was hitting the spot again.

1st International Workshop on Traceability

28 September 2002, Heriot-Watt University, Riccarton, Edinburgh

Andrea Zisman welcomed us to this first workshop, thanking the program committee for their hard work and Telelogic for its sponsorship.

Daniel Gross (University of Toronto) talked about designing systems (using any chosen notation) using ‘aspects’ to focus on quality requirements (usually global NFRs, eg performance, reusability, reliability, cost): how they are traded-off and refined during design.

In the example, he wrote a frame engine (querying, storing, and retrieving frames) language, Telos, in Prolog, to form a knowledge base. The rest of the work was at the meta-level, reasoning about the design choices to be made.

An Aspect is a new abstraction mechanism that captures design decisions that affect many modules. E.g. the choice of data passsing mechanism affects performance and reliability, so it is an aspect.

Both aspects and code modules are composed during implementation, affecting qualities in various ways. Traces can be used to link NFRs to design decisions and then on to system artefacts; the purpose, of course, is to orient system design to the various system goals, including NFRs – which people often more or less ignore when tracing back to functional requirements.

Intentional Agents can represent design goals; in the I* notation, that’s a big circle, which you decompose into tasks and goals with interactions such as ‘helps’ or ‘hurts’, effectively describing the reasoning underlying an Aspect of the system (like the data passing mechanism) according to what the Aspect ‘wants to do’ – in that anthropomorphic sense it is Intentional. The Agents are linked to the various Aspects that they generate (i.e. want). That stuff argues out the design rationale; it is then linked to the relevant design elements (code modules), by what I’d call a justification trace but which they call a reference.

Elena Perez wondered how this might scale up. Gross thought the representation was very compact; also you could have a folding editor that showed relevant detail and hid the rest.

Jeremy Dick and Ian Alexander said that one design choice often forced another. Gross said that you could use dependency links between tasks that the designers had to accomplish, to represent any kind of reasoning about the design.

Antje von Knethen (Fraunhofer Institute) spoke about automatic change support based on a trace model, to enable proper impact analysis before going for a possibly costly change. Documentation Entities talked about goals or requirements; Logical Entities spoke about the corresponding control tasks and environmental variables managed by those tasks. Actors could be linked to both. The relationships between entities include representation, dependency, and refinement. Then you can define constraints, e.g. in an embedded control system, each control task must ‘influence’ exactly one environmental item (like temperature). Obviously this allows you to carry out helpful validation checks on the installed traceability. A controlled experiment with graduate students showed that guidelines for maintainers using this approach were beneficial for impact analysis. The approach was therefore implemented by extending the commercial StP/UML tool. This was easy, and ensures the approach can scale, but the tool was unable to automate the making of changes or support the whole of the process – e.g. it didn’t support use case descriptions (and the use cases appeared as rectangles with no actor symbols!).

Darijus Strašunkas (Norwegian UST, Trondheim) spoke about traceability in collaborative systems development from a lifecycle perspective. Its purpose was to ensure completeness, to propagate changes, and to facilitate mutual understanding and enable meanings to be shared (‘semantic interoperability’!).

The essential difficulties included the distance between CP Snow’s two worlds – human usage/natural language, vs technical usage/formal methods. An environment was needed to support collaborative development.

Jeremy Dick (Telelogic) spoke about Rich Traceability. He said that we as a community were changing the world, at least because Traceability isn’t in any dictionary. We’re trying to get people to use it in real projects, not just in software but in systems of all kinds, such as the Joint Strike Fighter – probably the largest system development in the world.

What we say to our customers is that ‘information traceability is understanding how high-level objectives are transformed into low-level goals’, i.e. we abstract away completely from talk of links. And we tell them ‘the benefits include greater confidence that requirements are being met’ and that it gives you ‘ability to manage change’ and ‘improve collaboration between customers and suppliers’. This is important – we break down that barrier where people used to post a document to their customer, full of requirements: now the customer helps to write them. And finally it helps to ‘track progress and status’ and ‘to match cost against benefit’.

Rich traceability deals with and/or goal trees, to capture the ‘How do you know that this satisfies that requirement?’ of a problem. You can do the same for a validation strategy. Goal trees are not new – they’re used in van Lamsweerde’s KAOS, in safety case notations like Tim Kelly’s GSN, in Reveal’s satisfaction arguments, and so on.

Plain traceability just has N:M satisfies links, permitting impact analysis – if I change that, I better revisit those. But the bundle of links carries very little information about the satisfaction argument. Adding Assumptions helps; so does making explicit that a node is supported by ANDed or ORed requirements (visually, with ‘&’ and ‘or’ labels). Adding Arguments and ‘xor’ gives still more expressive power. These are implemented in DOORS – a simple way is to use attributes, and a more complete way is to use ‘establishes’ (1:1) and ‘contributes to’ (N:M) relationships, which can be displayed in a table-like view showing requirements, and/or, satisfaction, and contributing requirements alongside each other. An explorer tool allows arguments to be navigated and examined. This approach captures rationale, exposes it to peer review (which is critical for quality) and gives greater confidence in meeting objectives. Most new DOORS customers in the UK will use rich traceability at least in a simplified way.

I**an Alexander** (Scenario Plus) spoke about moving towards automatic traceability in industrial practice. He described the use of three simple tools (Scenario Plus add-ons for DOORS) in a train control box project: a use case linker, a dictionary term linker, and an exporter to create a hypertext from a use case model.

The effect of these is to speed and simplify the creation of requirements and to build a shared understanding (via the scenarios, and the agreed definition of technical terms) of the system approach. He illustrated how these worked in practice, arguing that having a set of documented stories and terms was a step-change improvement in industry.

The hard remaining task is to find ways of automating the linking of stories to requirements (argued Jeremy Dick): linking scenarios to tests is simple semi-automatically (with Scenario Plus) as testers can select and compose chunks of test cases very quickly using the existing tools. In a well-ordered project starting from scenarios, it is possible to copy-and-link to initialize the functional requirements with 1:1 links; if as in the railway project, the requirements already exist (in part), manual linking is today the only possibility.

Steve Riddle (University of Newcastle) spoke about enabling traceability. Current tools like DOORS and RTM could help but industry was (as previous speakers had mentioned) reluctant to move from paper-based methods. Things have moved on little in the decade since Olly Gotel and Anthony Finkelstein

'identified the crux of the traceability problem as "the inability to locate and access the sources of requirements and [pre-requirements specification] work"; they contended that this was a major contributor to the other classic (and still current) issues: out-of-date requirements; slow realisation of change; and poor reuse.'

This is indeed 'dispiriting', but it shows that Traceability workshops have something important to achieve. Semi-automation is perhaps what we need – e.g. identifying what needs to be updated, discovering 'implicit' traces buried in documents, and so on, much as Jeremy Dick and Ian Alexander had suggested.

Research needs to focus on technology transfer, to be pragmatic, and to address the issues by staying in touch with industrial sponsors, developing prototype tools to be evaluated by systems engineers.

Stewart Higgins (Philips) said that this presented traceability as a tools problem, but in much of industry the issue was the process in which such tools might be used; it was about people and what they did. Tools alone would always fail.

Patricio Letelier (University of Valencia) spoke on a framework for traceability in UML-based projects. Each project needs its own traceability approach, with no consensus on what information to collect, or what it means. Current tools focus on making a traceability graph between pieces of text. Integration between requirements and software tools isn't ideal; e.g. tools such as ReqPro don't offer a framework for traceability – all there is, is support for use cases and documents.

However a generalized framework was possible, e.g. stakeholder isResponsibleFor artefact; feature tracesTo use case; component isVerifiedBy test case. Generalized artefacts can be very few: traceable specification, model, and test specification. These can then be applied by specialization to a particular project, e.g. a RUP project can have use case model, class model, and so on as instances of model. Work is now in progress to build a module supporting the configuration of traceability for Rational Rose. Obviously this is aimed at software projects.

Alexander Egyed (Teknowledge) spoke on reasoning about trace dependencies in a multi-dimensional space. "I see traces as the simplest and most trivial artefacts of all, mere arrows, things that point at other things."

(Hmm, but what about relationships between requirements? Could be dependency, or constraint, or conflict, etc: would these work transitively, wondered Andrea Zisman and others of us. If transitivity didn't hold, the approach wouldn't work. In any case it only applied to software because everything related to the code.)

There are lots of manual tools and techniques: it's essentially a manual process, and you have to worry about properties like precision, completeness, and being systematic. We'd like it to be automatic! Hardly anybody really puts in traceability.

Traces are transitive and bidirectional, and therefore we can deduce commonality, i.e. if A and B each trace to C, A and B trace to each other. And if A and B trace to Ca and Cb, A and B trace to each other if Ca and Cb overlap.

For instance, if you make a change to requirement A, it will affect Code C and (therefore) may well have an impact on the ability of requirement B to get what it wants. With a trace analyzer, you can follow through any depth of tracing to identify all the trace dependencies between requirements no matter how distant the relationship.

If you have test scenarios, for instance, you can use a trace analyzer to find out all the classes and methods (and even lines of code) it uses and therefore verifies, and then automatically identify trace dependencies between code and model elements (assuming you know the relationship between scenarios and models). You can reason about the connections – via the overlaps in the code! – between dataflows, classes, and use cases, which constitute different dimensions of the problem.

We wanted, said Egyed, certainty about correctness and complete relationships between models and code. We get some way towards this. It is a simple approach that works, provided only that model elements are clearly distinct (which isn't always so).

Elena Perez-Mi•ana (Philips Laboratories) spoke about issues on the composability of requirements specifications for a family of products (TVs) in consumer electronics. These are still managed today through documents. There are incompatible standards for TVs in the USA, Europe and Asia. The configuration of specifications is hard to manage as there is a matrix of applicability against competences. Currently, researchers all use several sorts of model to describe product family requirements, and we concur. A database approach allows different views of a common set of requirements. A Domain Object Model (a class diagram) defines shared concepts with agreed names. A Use Case model defines the functional requirements for the Product Family. Use Cases are much more stable than the User Interface and other details of individual products.

We think use cases (she said) are the most effective way of defining product requirements, but in abstracting up to family level we still want to be able to see individual product details; and to see points of variation between products – both of individual features and of interactions between features.

We want to identify the right amount of natural language for product family specifications, as we think that informality leads to more inconsistencies, but readability is essential. We need a semi-formal language to allow us to express how products vary (as Mike Mannion has described). And we need to express temporal constraints, whether through UML activity diagrams or a semi-formal language.

The meeting ended with plenty of time for discussion. George Spanoudakis introduced the session with a 'biassed' list of open issues:

- Types of relations required – project/enterprise specific? Domain specific? With generic ontology? (e.g. impact traces (satisfies, verifies) vs others (justifies, is defined by)); Across all artefacts?
- Ways to automate the generation of relations: feasible? Ambiguity? Correctness & completeness – what can we tolerate? Quality assurance mechanisms? Can you create traces without automation, and can you trust them with it?
- Process issues: changes needed in human management? enforced while you develop specifications? Granularity of traceability? Or a posteriori, given the artefacts already? Traceability to product family specifications?
- Tool issues: interoperability and heterogeneity (in a distributed environment)?

We argued about much of all this. We didn't like the Gotel/Finkelstein Pre/Post-traceability distinction – it's all the same stuff, we felt.

Jeremy Dick suggested that all true traceability relationships are many-to-many.

Installation of traces is far more difficult on brown-field, legacy system projects; and automation would be valuable, if only it generated trustably correct links.

Andrea Zisman defined Recall as # of detected links / # of possible links; whereas Precision was # of correctly detected links / # of detected links. A precision of 80% is, we felt, like an Internet translation of a foreign document: 20% incorrect can be really distracting, though we may be able to correct wrong links; on the other hand, missing links are a real problem.

Traceability is vital for change management and impact analysis; but it is not sufficient for this, as it says nothing about new requirements. And a simple 'suspect links' flag can indicate only that an item might need to be changed; not that anything that matters has occurred. Wim Dekker suggested that a traceability link is one that cannot be automated! Alexander Egyed said that if you wanted traces from models to be accurate, then you had to have a fine grain. Coarse links would be inaccurate. Stewart Higgins said that inaccurate traces would lead to expense as they'd need to be checked by hand. Alexander Egyed said that existing automated methods could do good and useful work already, saving money.

We concluded that Ariadne, who gave the hero Theseus a reel of thread to find his way back from the Minotaur's labyrinth, was the founder of Traceability.

RESG Security Risk Analysis & Management

Imperial College, 29 Jan 2003

Bashar Nuseibeh (Chairman) welcomed about 40 people to Imperial College on a cold windy winter's day for our experimental format event, a masterclass on Security. Security was sometimes glossed over as a 'Non-Functional Requirement' but it was unusual in being negative – the stakeholders we really wanted to interview and who threatened our systems were not available for comment!

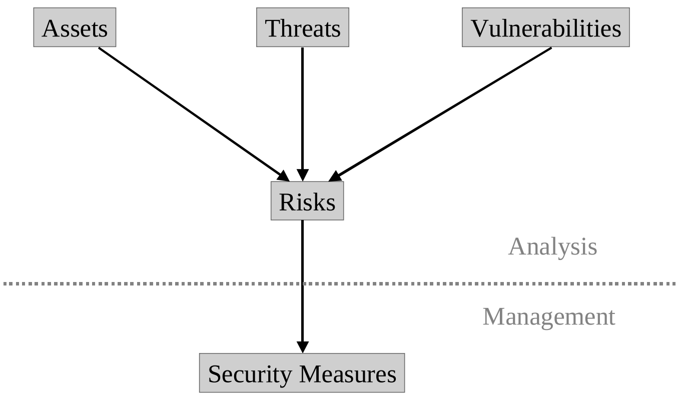
Jonathan Moffett ('**JM**') courageously offered his expertise and **Francesco Arci** ('**FA**') acted as the client.

JM said he was very happy to be on this Blind Date with FA, and introduced his **risk analysis method** as follows. Baskerville (1993) had proposed 3 methods: basic checklists; engineering mechanisms such as risk analysis; and finally integrated design – which no-one has yet achieved. Quite a lot of requirements assumptions didn't work in security; and security people often didn't know what security tasks really belonged in the early requirements phases.

Security analysis looks at the assets to be protected; the threats to and vulnerabilities of those assets, and hence the risks to be mitigated; and thus the security measures to be taken in security management (see diagram). JM defined the following terms:

Asset: hardware, software, data, people, documents, goods, money, intangibles.
Threat: type of harm that can happen to an Asset
Impact: measure of seriousness of a Threat
Attack: a Threatening event
Attacker: agent (human or not) causing an Attack
Vulnerability: weakness in the system that makes an Attack more likely to succeed.

Risk analysis had as goals identifying all assets, all threats, and all vulnerabilities, as well as valuing the impact of the threats and assessing the vulnerabilities. Value can't always be in money – data, in-house software, goodwill, and customer confidence aren't easily priced, yet their loss can threaten bankruptcy. Past events such as fires in computer rooms (10^{-4} per room per year) may not predict future ones as attackers may vary their behaviour – since Al Qaeda appeared, our view on what threats were credible has changed. So maybe high-medium-low or 1..10 may be better than precise money values. Maybe a £10,000 loss is Medium impact for a typical university department, for instance.



Francesco Arci agreed £100,000 for medium and £10M for high in his case.

JM said the steps were to decide scope of analysis (boundary on context diagram, and looking at security of neighbouring domains); identifying assets and threats and valuing them (all of this was RE); identifying and assessing vulnerabilities to threats, and risk assessment – these arguably being part of design not requirements.

Threats could include loss of confidentiality, integrity, availability or simply of money. Some business processes are also critical – even the loss of a cheap computer could be serious if a major process is disrupted.

Vulnerabilities could be of any kind; good practices to identify them were listed in BS 7799. We could then assess the likelihood of people attempting them, and of succeeding: vulnerability is a combination of these two factors. E.g. with ADSL, people constantly attempt to break into JM's PC, so the probability is 1; but with a firewall the chance of success is low; the resulting vulnerability looks to be Medium, but this is a purely intuitive calculus (JM uses a simple matrix). You could in principle compute risk but JM doubts it is practical to do so; a simple matrix (lookup table) again seems the best approach.

When people know of risks they can ignore them, try to transfer the risk (e.g. by insurance), reduce vulnerability (e.g. firewall), reduce impact (e.g. fire extinguisher), or prepare recovery measures (e.g. restore from archive). Risk management means identifying and selecting appropriate security measures commensurate with risk.

Adding security measures changes the system; e.g. encrypting all exam papers removes a threat to Confidentiality, but if staff forget their encryption keys then it introduces an Availability threat.

Francesco Arci of JP Morgan Fleming Asset Management described the retail asset management **problem** to be evaluated today. The company sells services to the public in Europe, either via other brokers or advisers, or direct to clients. This means having to authenticate the client or broker; e.g. on a voice line this is via a set of questions on a workflow system, which is expensive in people's time. Different procedures apply to voice, fax, PDA and Internet access. The company would like to enable a virtual call centre where location of the 2 call centres is unimportant (and they can cover for each over, and share skills); and to move more of the business to self-service rather than via human agents. OCR of paper documents might help but only as an interim step.

Secrecy laws in Luxembourg and Switzerland are different; regulatory constraints limit who can see what data (in which country); it is better to allow only authorised users to view specific data items; and decryption of some items is also restricted geographically according to the applicable jurisdiction. Traditional restrictions on client confidentiality and non-repudiation apply.

JM then started to **elicit the requirements** from FA. There was an authentication machine, separated by a 'DeMilitarized Zone' (DMZ) from people outside ('external parties'). FA wanted to unify the authentication mechanisms, and do it consistently. If this was infeasible then FA wanted to iterate with (next time around) a multi-mechanism approach. JM wondered if Usability might not make unification infeasible – computer users could accept complexity that phone users often did not.

The authentication machine controlled access to the services, which shared various internal storage devices. Regulation demanded an audit trail, which was also useful internally. Someone controlled registration under a process for checking credit limits. Could this be purely electronic or did there need to be a contact with the real world, asked JM. Misuse needed a human decision to deregister people; companies often wrote to people to tell them their access codes rather than risk relying on pure computer access. FA said the country-specific items had to be segregated, so JM drew 2 stickmen (maybe swiss and luxembourgish admin) to control registration. FA said internal users also had to be authenticated (the same way), with access rights according to their group. JM built up an agent interaction diagram with boxes and stickmen. JM assumed bad behaviour on the part of both internal users and external parties, but would rely on good behaviour of administrators, so he drew them inside an authentication system boundary that also included the authentication machine.

JM pointed out that merely analysing the security of the authentication machine would be inadequate – nearly all security breaches (see Kevin Mitnick's *Art of Deception*, said JM) involved both human and machine weaknesses. Nearly all Mitnick's best hacks, he said, involved talking to 3 people and putting together the whole picture: misuse cases might be a good idea but only dealing with one threat at a time might be a contributory cause of successful hacking, so it was all Ivar Jacobson's fault.

JM deleted the DMZ from the diagram; Richard Veryard said that that meant it was a neighbouring domain that was trusted, but that maybe it should still be left in the picture in some way. JM said that the needs of security analysis here were more restricted than that of a full requirements analysis.

JM asked if we were just authenticating, or if it was more than that – as someone asked from the back, was Authorisation also involved? FA said yes; they needed to replace the old mechanisms and also put in place wholly new functions. JM said that things like Encryption were probably design decisions, but FA said there were requirements there, e.g. the UK operator must not be able to obtain decryption keys, lest a UK judge granted an injunction to obtain the keys and hence breach Luxembourg law. There were several similar (and complex) requirements influenced by legal considerations. JM checked that FA felt it was hard to disentangle the 3 main functional areas; JM described them as **authentication** (verifying a claimed human identity); **authorisation** (explicit); **encryption** (an implicit sort of authorisation, perhaps). FA added **non-repudiation** as a requirement also. That completed JM's first stage of analysis.

Assets included intangibles like reputation – its loss was a highly important threat, as it influenced the share price. JM added ("this is cheating") that 'the process' was also an asset. One of the unintended consequences of automation is removing fallback positions that you had before – if you no longer have phone and fax your business is less resilient if your computer fails than before, said JM. A 2-hour outage was a medium impact risk; a 1-day outage was a high impact risk, said FA; a half-hour outage was the most expected by the business. Failure of integrity was also high (even if FA wouldn't admit that it might be caused intentionally by competitors).

The discussion was then opened to **the audience**. Stephen Murgatroyd asked what authorisation was? What was a user? What was security? Hubert Matthews asked what was 'good enough' for authentication? Biometrics? DNA? People had different expectations, especially wrt different channels. FA agreed; all trades were captured and verified so there was a fallback. A PIN plus an account number would probably be acceptable for an automated trade; usability had to be traded-off against security. JM said that there were FSA guidelines on what was good enough; and wasn't it just like the elicitation of any other non-functional requirements?

Howard Chivers asked if JM shouldn't have questioned further about the proposed solution approach. JM said it would have been impertinent and a recipe for disaster to say to the client 'oh, you don't want authentication'.

Bashar Nuseibeh mentioned James Robertson's advice to analysts to 'invent requirements' (in *IEEE Software*, July/August 2002); how far was it appropriate to suggest things to clients? JM said that authentication was an entirely legitimate requirement within the scope; we'd moved down a level from 'we want to make profit' and that was perfectly reasonable. FA thought the questions were at the right level, given the time available.

Ian Strong wondered if one might monitor behaviours, as Visa notice strange patterns like large overseas transactions. FA said yes, they did that in the Workflow which had a rules engine looking for such patterns. JM said if we'd had time we'd have considered certain vulnerabilities and then security measures, including detective controls such as looking at monetary behaviour… in a couple of hours' time.

Stephen Murgatroyd asked if there were metrics on security breaches? FA said no, as the channels were separate; there were metrics on efficiency and downtime, however. JM said he'd be politically incorrect and say that if the figures were there, fine, but if not and he could improve security by issuing passwords at £5 a time and that'd be cheaper than doing the analysis then he'd just go right ahead with the action.

Howard Chivers said many such requirements came out of audits. Although regulators and auditors sometimes specified a particular solution to a security requirement, they would often accept any reasonable alternative. Richard Veryard said that this indicated that such requirements could regarded as negotiable – but that didn't stop them being requirements.

JM said he was disappointed even though it wasn't FA's fault that we'd had to analyse security requirements of a security system – which was bound to muddy the waters. FA said that unfortunately that was what they needed.

Richard Veryard wondered if JM could cover things like active threats within your system. JM said like Back Orifice (!) which planted itself in your computer and could later be used for further attacks? Veryard said yes, or like planting a human mole in your system administration. JM said you'd never guarantee to think of all future threats. Hubert Matthews said it was a pity that the security system had been assumed to be almost a bolt-on to the rest of the company – wasn't security pervasive?

Stephen Murgatroyd said that the main problem we'd had today was that the system scope wasn't fully defined so we didn't know what we ought to be looking at and kept on coming back to whether there should be a DMZ, etc.

JM said that in vulnerability analysis you might find you have the wrong model – too many or too few functions and hence you'd then have to iterate.

Bashar Nuseibeh said while we couldn't necessarily draw conclusions, there were several threads of questioning that would improve both the security and the broader requirements process, and that were valid subjects for research.

The audience had clearly enjoyed this novel event but would have liked more, which is encouraging; on the other hand, it was a taxing time under the spotlights for the presenters, and the RESG is very grateful to them.

Richard Veryard adds …

I enjoyed this experiment at several levels, which made a number of security requirements issues visible for me. I also came away with a number of questions about how the process would work for more complex sociotechnical systems.

Stakeholder. Normally, the concept of "stakeholder" or "interested party" implies some degree of legitimacy – and a legitimate stakeholding may carry a moral or procedural right to be consulted or represented. Any individual can claim a stake in any problem, but only becomes a true stakeholder if this claim is recognized as valid. Including hackers as stakeholders introduces some complex scoping and ethical issues, which were not addressed in this meeting.

Assets. JM's process regards asset as a key starting point for the security analysis. This obviously raises questions of scope – whose assets? A security analysis for a financial company may obviously need to consider the security of its customers' assets as well as its own – and this should probably include intangible assets such as privacy and confidentiality.

> *He that filches from me my good name Robs me of that which not enriches him, And makes me poor indeed [Othello]*

It seems to me that there are two main security reasons for caring about assets. One is that assets represent value in their own right – and therefore any loss or damage to an asset affects the balance sheet of the enterprise. Loss or damage to third party assets may be just as significant as loss or damage to owned assets, since this may well generate liabilities. If assets are in the method because they represent value, then it makes sense to include liabilities as well. Conversely, if owned assets are adequately insured, so that losses are covered by a third party, then their security could possibly be regarded as someone else's problem. However, it is dangerous to rely too heavily on insurance companies, since insurance can sometimes become unavailable or unaffordable at short notice.

The other reason for caring about assets is that the business process may depend upon them. If a train derails and chews up the track, this is a problem for the train company for several reasons. It is not just because the train company may be financially liable for the damage to the track, but also because the train company will itself suffer from the unavailability of the track while it is being repaired.

The impact of loss or damage to a small asset, or the consequent damage to other assets, can be vastly more than the value of the asset itself. In industrial accidents, this is clearly understood. The damage that can be done by an asset is often far greater than the value of the asset itself. Under certain conditions, a trusted employee such as a rogue trader or creative CFO can wreak incalculable damage. Terrorist attacks often use this gearing to great effect. A car filled with explosives can wreck a shopping centre; a few knives on an aeroplane can destroy an office block.

Reputation, and a good relationship with the regulator, are therefore assets in this sense. A security breach may prompt the regulator to impose additional security measures; a change in status may mean that the enterprise has to allocate greater resources to contingency reserves.

In the meeting, JM said that it was "cheating" to regard the process as an asset. I agree, and I prefer to regard the business process as something important in its own right, with critical dependencies on key assets.

Threat. Damage to the business can come from within. If we regard an unalterable audit log as a security mechanism, then this is an implicit recognition of a security requirement that references the possibility of harmful actions by insiders – and this may range from carefully planned fraud to panicky attempts to hide one's mistakes.

Complex threats may depend on a combination of factors, some internal and some external. Kodak lost a lot of money as a result of customers, prompted by a third party website, accessing a page on its e-commerce website with an incorrect price for a digital camera. The analysis of such threats is not amenable to a simplistic topology of inside/outside.

Vulnerability. In normal speech, we use the term vulnerability in two distinctly different ways. People are vulnerable to bullying or stress; organizations are vulnerable to fraud. Employees can be amenable to blackmail or corruption; employment practices may be unable to detect recruits with criminal intent. When discussing social engineering attacks, therefore, we may describe people and organizations as being in a state of greater or lesser vulnerability.

However, when discussing the security of computer hardware and software, the term vulnerability always implies the existence of some attack mechanism. Buffer overflows represent a general vulnerability because hackers know how to exploit them. Vulnerabilities are now discrete countable things, which can be specifically located within the system. Each vulnerability belongs to a specific component, and computer security becomes an amusing game of blame–the–component.

Incidentally, Microsoft's Trustworthy Computing initiative can be interpreted as an attempt to get away from the game of blame–the–component, towards an ecological way of thinking about trust.

A **strategic attack** can create new vulnerabilities in this sense – whether planting a mole in the organization, loading malware such as Back Orifice onto a computer, or even hacking into the vendor's development environment and placing security holes in the next version of the software. Such an attack may cause no direct damage, and remain undetected for years.

An **autonomic system** should have the properties of protecting itself from a defined range of attacks, including strategic attacks. This raises the interesting question: how do we define the requirements on an autonomic system? At one level of course, autonomic computing can be regarded merely as a superior solution to a given set of security and other requirements. But it is not hard to see how this leads to a series of lower level requirements, which include the specification of autonomic behaviour.

How then does security requirements engineering differ from regular requirements engineering? Perhaps in future not so much. Dick Baskerville's work ten years ago showed the way towards an integrated design method – and I hope that RESG mounts further experiments of this kind to push practice forward.

RE-Papers

Requirements, Myths and Magic

Ian Alexander
Independent Consultant

Saying what we want isn't new; people have wished for a better life since the dawn of time. Western culture has thrown up several myths about asking for and getting what you want; but all of them are rich in warnings which may be salutary reminders. Let's just look at three of them.

One of the Greek Myths of asking and getting is the sad tale of Pandora, whose name means 'all-giving', itself a warning. She was created by Zeus when he discovered that Prometheus had stolen fire from the gods, and was hence able to do many creative and destructive things that had until then been divine privileges. Men 1, Gods 0. Zeus at once ordered Hephaestos, the smith of the gods, to create the image of a beautiful maiden, Pandora. Athene dressed the treacherous thing with lovely clothes. Hermes filled a vessel to accompany it with all the evils and troubles the gods could devise. All the mortals and immortals were astounded by Zeus' ingenious answer to man's attempt at becoming equal in power to the gods: for here was the start of the race of women, "that threatening wile against which men are defenceless"[1]. Men 1, Gods 1. (Political Correctness, 0.)

Worse was to come. Prometheus (whose name means 'Provident') warned his brother, Epimetheus ('Heedless'), not to accept any gift from the gods. Zeus sends Pandora to Epimetheus as a gift, and – true to form – Epimetheus accepts. Pandora opens the vessel and all the evils escape and spread throughout the world, leaving only Hope behind. Gods 2, Men 1.

Pandora's vessel came with a prohibition – it was not to be opened (just as the fruit of the tree of knowledge in Genesis was not to be tasted). We've all seen projects that went hell-or-high-water for new technology without too much cautious work on discovering requirements or evaluating risks, and all the evils – delay, cost overrun, blame, slashed functionality, bugridden software – sprang from Pandora's box and could indeed not be put back again. In the myth, one of the evils – Hope – stays behind in the vessel: it's the sting in the tail, as the project manager continues to produce hopefully optimistic plans that show that all is well while everybody knows the project is sliding helplessly to the right, thus guaranteeing continued

[1] C. Kerenyi, *The Gods of the Greeks*, 1951; Thames & Hudson paperback edition 1974.

anguish while the watching Gods chuckle. One form of this hope is that a Method or Tool (or even a Famous Consultant) can be found to bottle the spirits, to slay the demons with a magic silver bullet, to ward off the forces of evil with an amulet. The unscrupulous will always be on hand to sell Amulet-ware (version 5.1).

Perhaps the best-loved story of specification is Pygmalion. Who? Pygmalion is ancient Greek for Dwarf or Pygmy, i.e. the short and ugly fellow who couldn't get a girlfriend. Being clever and skilful he decides to make one instead, and carves a beautiful woman in the finest white marble from the island of Paros. At once he falls in love with the statue, and longs for the gods to take pity on him and breathe life into it. At last they relent, and she comes to life as the perfect nymph; the only problem is that she isn't particularly keen on the Pygmy himself. Bernard Shaw's play (also called Pygmalion) has an English gentleman breathing elegance and elocution into a market-girl, with the same result as the Greek myth, retold (again) as My Fair Lady. Yet another variant on the theme is Mary Shelley's Frankenstein, where the mad inventor succeeds in playing God, only to find that his long-sought creation becomes a monster. And, yes, we can all think of projects where the requirements seemed to be all right at first, but… Pygmalion is the definitive myth of technology, computing, robotics, artificial intelligence, genetic engineering and all other attempts to specify better systems. Your system may work perfectly – but not deliver the results the users wanted or expected. The enduring popularity of this myth says something about public attitudes over the millennia to technology and technologists.

Finally, and most directly for requirements people, there is the folktale of the Three Wishes. There are any number of versions, notably including Saki's; the genre is mentioned (but not retold) by Gause & Weinberg[2]. (Readers of delicate disposition are advised to skip to the last paragraph now.)

A poor soul wishes desperately for better things. In a tavern he overhears a conversation about the Monkey's Paw and is drawn unsuspectingly in. A strangely-dressed man explains that the thing allows its bearer to obtain whatever he wishes for! Visions of wealth and luxury and eternal youth float before the poor man's eyes. He manages to obtain the amulet and makes his first Wish – a great treasure of 100 gold crowns. The very next day a messenger dressed in royal attire appears at his tiny hovel, announcing that with deepest regret they have to inform him that his son has been killed working for the king, who has sent a present of 100 crowns to mark his feelings.

Horrified, and aghast at what he has done, the poor rich man wishes his son was alive again. The very next day, a sister arrives from the hospital and tells the man that his son is alive, but still terribly mutilated, unable to help himself, and that if he can care for the patient he can come and collect his son. He does so, and he and his wife work hard every day looking after their living-dead boy. Things are so heartbreakingly bad that eventually the sad man wishes silently to himself that his son were dead again, and at once the boy dies. The stranger from the tavern who had been talking about the wonders of the Monkey's Paw appears from nowhere, and asks if the man was happy with his Three Wishes. The man speechlessly returns the hated amulet to the stranger.

All of these myths agree that it is indeed possible to specify systems of great power, and to have them built; but that getting what you really want is another matter entirely. Have you ever tried to frame your Three Wishes, if you could have anything you asked for? It isn't easy to specify your requirements.

Generation Of Requirements From Scenarios

Janos Korn
London School of Economics

Background

At the IEE colloquium 'Scenarios through the systems life cycle', 7the Dec 2000, I presented a brief paper titled 'Scenarios through linguistic modelling'. Later I talked to colleagues whom I had met at the colloquium and consulted some books and journals on the subject of 'requirements'. I found that there was a certain lack of consensus about what was meant by the term 'requirement' in a design context. Also, writers used a variety of methods like data flow diagrams and systems dynamics borrowed from systems science for modelling scenarios. UML diagrams and use cases although more comprehensive, appeared to follow a similar line. A great deal of resources seemed to have gone into software development. These methods have superficial empirical/theoretical foundation, they have no basis in existing branches of knowledge, make no explicit reference to changes of state in time and their use of 'properties' is scarce. There did not seem to be a clear explanation of how requirements could be deduced from scenarios. The general opinion appeared to be that requirements were elicited from a client or a user.

The problem was then to develop a design method which would show how to generate requirements with a methodical representation of scenarios as an integral part. I have published a number of papers dealing with representation by means of linguistic modelling (LM) which does not seem to have the problems briefly referred to above. The objective of this short and fairly informal presentation is to summarise the current state of 'generation of requirements from scenarios'.

[2] *Exploring Requirements*, Dorset House, 1989.

Features of the proposed method

Features of the method of generation of requirements here introduced, are :

1. Representation of a scenario by LM begins with a story or a narrative describing some activities which is then subjected to *linguistic analysis* to convert linguistic complexities into a homogeneous language of one- and two-place, simple sentences which are then sorted into sentences with energetic and informatic interactions. These are represented as a diagram which displays the *topology* of a scenario from which series of predicate logic forms can be deduced. These forms can carry *uncertainties* associated with behaviour of human and/or other kinds of components together with computations as needed. The logic forms represent sentences which express interactions as *skilled power* (with appropriate energy) or *influence* (carrying information) through dynamic verbs like 'to drill' (a hole) or 'to notify' (the public that….). The diagram shows explicitly the outcomes of activities in a scenario as properties when progressions of states *in time* terminate.

2. Requirements are seen to originate from properties which describe the outcomes displayed by an '*initial representation*' of a scenario. These outcomes are considered unsatisfactory and are seen to need specific changes.

3. We introduce the notions of '*dynamic' and 'stative'* requirements. The former is needed to describe a cause of a '*changing property'* of a changing object, the latter are derived from *'quiescent properties'* which are relevant to change and accompany a changing property. *Getting from one place to another* may require a vehicle, in particular a taxi, but the *number of passengers* determines its size.

4. The *first dynamic requirement* calls for the cause of a changing property and points to the selection of one of the *'generalised products'* : artifact or energy/medium for 'physical' or information/medium for 'mental' change of state. Product is needed to generate the interaction or process to induce the change of property in a chosen changing object. The *second dynamic requirement* calls for the delivery of product to the changing object by 'interacting objects' (IO) operating purposively and points to the selection of IO. The *stative requirements* are the consequences of quiescent properties and point to the realisation of specific properties which supplement the changing property.

5. The emerging product and IO are inserted into the initial representation of a scenario leading to a *'subsequent representation'*. This latter appears as a diagram which can be subjected to the same kind of analysis as the initial or any other representation of a scenario by LM. *A uniform approach has emerged.*

6. Properties of any product are described by declarative, simple sentences. Thus, their *effectiveness* can be worked out in a unified way whether a product is an artifact, energy or information.

An example

A. Initial representation of a scenario

Scenario : 'George thought that he was badly served in a restaurant. A decisive man, he immediately picked up the phone and complained sharply to the manager who answered the phone, that he had a long wait for food which felt cold when it finally arrived and its taste was poor. The manager was a sensitive man who preferred to see complaints in writing, with a sense of maintenance of quality of service and food : he expected a waiting time to be less than 10 min, the food to be around 25°C and its taste to score at least 7 out of 10. He considered the situation with a view to contact George'.

After linguistic analysis the scenario is diagrammed in Fig.1. with outcome 'manager with consideration of situation'(ap(5,5)). This may be considered unsatisfactory by 'george'. He can set his objective as 'to complain so as to get compensation from the manager'.

B. Subsequent representation of a scenario

Having considered the initial representation of a scenario, the design process is concerned with development of a subsequent scenario. We use the scheme in Fig.2. as a guide.

Quiescent properties are related to : changing property - CP, changing object - OP, environment - EP, alternatives – AP. Initial and final properties/conditions – IC, FC.

Analysis of change (An empirical exercise)

Changing object (a noun) – manager

IC (sentence with a stative verb) – Manager is unaware of complaint

FC (sentence with a stative verb) – Manager is aware of complaint with request for compensation

Changing property (sentence with a dynamic verb) – Manager becomes aware of complaint with request for compensation

Quiescent properties (sentences with stative verbs) –

CP,1 – complaints go directly to manager
OP,1 – manager is a sensitive man
OP,2 – manager prefers printed communication
OP,3 – manager's expectation for waiting time is to be less than 10 min
OP,4 – manager's expectation for food is to be around 25°C
OP,5 – manager's expectation for food is to score for taste at least 7 out of 10
EP3,1 – consumer association encourages customers to ask for reasonable compensation

OP,6 - manager prefers communication to be private

AP1,1 – manager prefers to see complaints in writing
AP2,1 – george has little confidence in anybody
EP3,2 – communications are carried by post

Scheme for use of changing and quiescent properties to generate requirements and to choose product and IO (A theoretical exercise)

1. **First dynamic requirement** (to indicate cause of change):
‘Sentence with changing property’- *requires* – ‘Sentence of cause with (*generic* initiating object + dynamic verb as infinitive complement)’

2. **Selection of product** (to affect choice of product): ‘Sentence of cause with (*generic* initiating object + dynamic verb as infinitive complement) AND (AP1, EP1 and 3)’ – *selects* – ‘Sentence with (*name* of product + dynamic verb as infinitive complement and its function expressed as adverbial)’

3. **Second dynamic requirement** (to indicate cause of delivery):
‘Sentence with (*name* of product + dynamic verb as infinitive complement and its function as adverbial)’ - *requires* – ‘Sentence describing delivery with (*name* of product + qualified, dynamic verb as infinitive complement’

4. **Selection of IO** (to affect choice of IO): ‘(Sentence describing delivery with (*name* of product + qualified, dynamic verb as infinitive complement) AND (AP2 and EP2 and 3)’ – *selects* – ‘Sentence with (*name* of IO + qualified, dynamic verb as infinitive complement)’.

5. **Stative requirements** (to generate particular properties of product and IO): ‘Sentences with quiescent properties’ – *require* – ‘Adverbial and adjectival sentences with (*specific means* + stative/dynamic verb as qualified, infinitive complement)’

Application of the scheme leads to :

Dynamic Requirements

1. 'Manager becomes aware of complaint with request for compensation' *requires* 'Means as encoded medium which is to convey complaint with request for compensation'

2. 'Means as encoded medium which is to convey complaint with request for compensation' AND ‘AP1,1 = (Manager preferred to see complaints in writing), EP3,1 = (Consumer association encourages customers to ask for reasonable compensation)’ *selects* 'Letter (medium : white paper) with information is to be carried to manager so as to convey the complaint with request for compensation'

3. 'Letter (medium : white paper) with information is to be carried to manager so as to convey the complaint with request for compensation' *requires* 'Letter (medium : white paper) with information is to be carried to manager'

4. 'Letter (medium : white paper) with information is to be carried to manager' AND ‘AP2,1 = (George has little confidence in anybody), EP3,2 = (Communications are carried by post)’ *selects* 'George is to deliver letter (medium : white paper) with information to a post box' (**dp(1,12**) in Fig.3.)

Stative requirements

5. Adverbial sentences related to product and IO for **ip(7,8)** and **ip(1,10)** in Fig.3.

CP,1 – 'Complaints go directly to manager' *requires* 'Letter (medium : white paper) with information is to be in his intray’
OP,1 – 'Manager is a sensitive man' *requires* 'Letter (medium : white paper) with information is to point out deficiencies in service and food politely'

Adjectival sentences related to product and IO for **dp(1,1), dp(1,10)** and **dp(1,11)** in Fig.3.

OP,2 – 'Manager preferred printed communication' *requires* ‘Letter (medium : white paper) with information is to be put in printer' (**dp(1,1**))
OP,3 – 'Manager's expectation for waiting time was less than 10 min'
OP,4 – 'Manager's expectation for food is to be around $25^{o}C$'
OP,5 – 'Manager's expectation for food is to score at least 7 out of 10 for taste'
EP3,1 – ‘Consumer association encourages customers to ask for reasonable compensation'
require

'Letter (medium : white paper) with information is to be written as complaint : (**dp(1,10**))

1. waiting time for food was 20 min,
2. food was lukewarm at $20^{o}C$,
3. score for taste of food was 5,
4. a free meal is serve as compensation,

OP,6 – 'Manager preferred communication to be private' *requires* 'Letter (medium : white paper) with information is to be put in an envelope' (**dp(1,11**))

Concluding points

1. We have outlined a design method embedded in LM which produces requirements leading to possibility of choice of product and IO. The method operates in terms of properties to which a particular product and a system can be fitted. The design procedure begins with changing and quiescent properties the identification of which is an empirical exercise.

2. A logical procedure originating from these properties has been shown. Involvement of a client or a user is essential as the agent with knowledge of a particular scenario.

3. Dynamic and stative requirements have been introduced. The use of the former is usual practice in engineering.

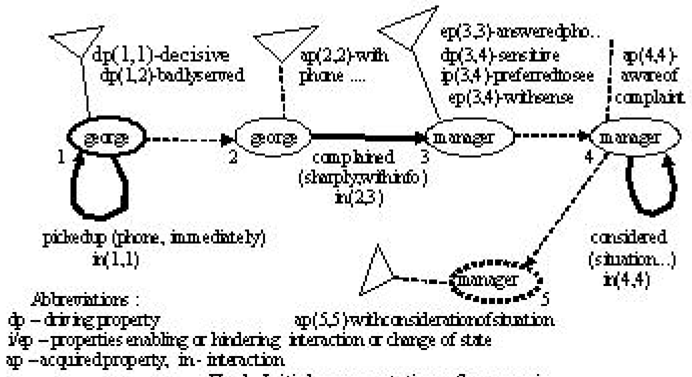
4. 1st and 2nd dynamic requirements expressed in generic terms to admit choice plus AP and EP properties are directed at selecting a product and IO, stative requirements refer to properties of product and IO once selected.

5. Properties of products (artifacts, energy, information) are expressed linguistically in a uniform way. Thus, their effectiveness can be evaluated uniformly (not included here).

6. The operation of the scheme in Fig.3. is governed by an 'objective'. This point has not been considered here. The manager's *decision process* whether to give compensation or not is not shown.

7. Carriers of EP properties bring into the scheme stakeholders and interested and affected parties.

8. The use of *generic* objects, AP and EP properties enables the entry of choice and creativity into the scheme.


dp(1,1)-decisive
dp(1,2)-badlyserved
ap(2,2)-with phone ....
ep(3,3)-answeredpho..
dp(3,4)-sensitive
ip(3,4)-preferredtosee
ep(3,4)-withsense
ap(4,4)-awareof complaint
george
1
george
2
manager
3
manager
4
complained (sharply,withinfo) in(2,3)
pickedup (phone, immediately) in(1,1)
considered (situation...) in(4,4)
manager
5
ap(5,5)withconsiderationofsituation
Abbreviations :
dp – driving property
i/ep – properties enabling or hindering interaction or change of state
ap – acquiredproperty, in- interaction
Fig.1. Initial representation of a scenario

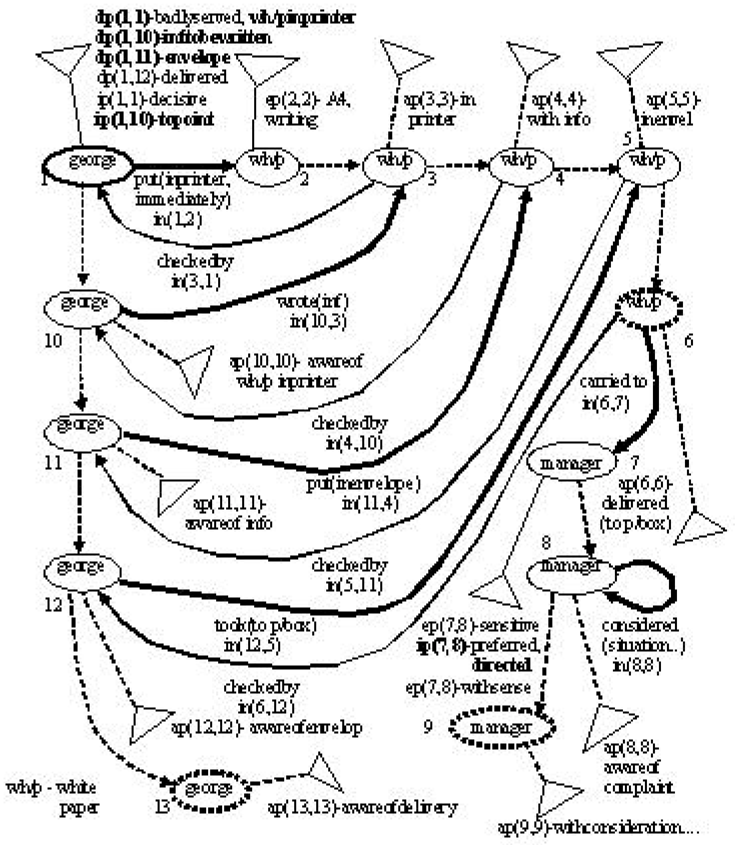


Fig.3. Subsequent representation of scenario

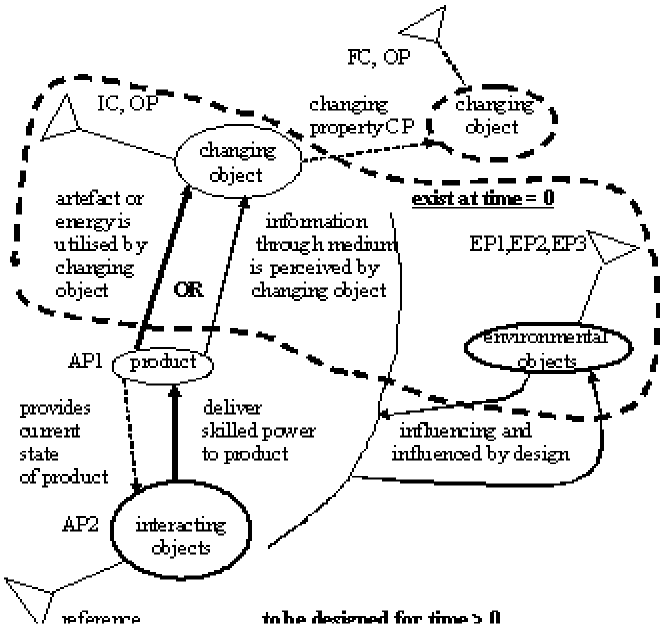

FC, OP
IC, OP
changing property CP
changing object
exist at time = 0
artefact or energy is utilised by changing object
OR
information through medium is perceived by changing object
EP1,EP2,EP3
environmental objects
AP1
product
provides current state of product
deliver skilled power to product
influencing and influenced by design
AP2
interacting objects
reference
to be designed for time > 0

Fig.2. Design scheme

RE-Publications

All reviews by Ian Alexander

Tell Me A Story

Roger C. Schank
Northwestern University Press, 1990
ISBN 0-810-11313-9

In this pioneering, enjoyable and insightful book, Schank leads the reader to start from scratch when thinking about what we mean by knowledge, memory, and intelligence.

His thesis is essentially that being human means telling stories:

"...all we know is embodied in stories."

We remember stories, and perhaps they are the key to how we organize our memories, which is to say our minds. We know what to do in a situation because we have a wired-in script that we have (maybe painfully) learnt – and we behave with skill in that situation because we can follow that story efficiently. We can intelligently plan and reason about alternative plans, because we can tell stories with happy or unhappy endings for each variation.

I hope this is enough context to make it quite evident that Schank is totally relevant to thinking about how to capture and organize requirements. His work on Scripts set the tone of much of the Artificial Intelligence research in the 1980's: what was then called "knowledge elicitation" has of course transmuted into "requirements elicitation" today (see 'A Retrospective on Knowledge Acquisition' below). Schank consistently argued that people ran their lives by stories, such as the famous restaurant script – you go in, hang up your coat, get shown to a table, sit down, read the menu and so on. This was taken terribly literally as a precise set of rules at the time, which (of course) didn't work; but Schank's work has also always emphasized the vague and associative nature of human thought, such as the way we connect one story with another, or events and details with a story. This should have warned people not to be too rule-bound and literal.

Schank writes fluently, cogently, and with a sense of fun as well as of exploration -- he dares to break the rules (and across the cover of the book is a strapline that says "rethinking theory"). The rethinking has to reach back far into the past, indeed to Aristotle, whose view of reasoning was that it had to be theoretical and hence rule-based, *episteme* – and for the last two millennia, the theory of knowledge has been Aristotelian "epistemology". A competing view even in ancient times was that you could reason from instances or cases, *casuistry*, which like rhetoric and sophistry was not originally a pejorative term. Only since the 1980's has case-based reasoning come to be honoured again, and rulebases have come to be seen as hopelessly limited except in very tightly-delimited domains. It's almost funny that it has taken 2000 years to get back and fix mistakes made by the Ancient Greeks, but that's the way it is. For me, this is extremely important – it is a revolution in thought, and it goes a long way to explaining why traditional arguments always break down. Another (closely-related) part of the repair and replacement of outworn arguments is Polanyi's demolition of the idea that knowledge is conscious – much of the way that skill works is plainly tacit.

Given the contributions of Polanyi and Schank, requirements engineers – indeed, system thinkers and developers in general -- need to come to grips with the fact that people, even if highly skilled and even 'expert', cannot give us precise rule-like requirements. If we try, we will get bad and wrong requirements. Instead, we urgently need to start thinking and working in stories – scenarios, use cases, scripts, user stories, whatever. We can use these as hooks to retrieve connected ideas and build a shared understanding of what people need, by talking about what they actually do. Then we'll have requirements that are somewhat closer to reality.

The book looks at where stories come from and why we tell them; how we understand and index stories; how stories shape memory; "story skeletons", i.e. the structure of stories; the stories of different cultures; and the role of stories in intelligence.

There is a very short list of references, but Schank sprinkles the text freely with quotations and jokes, examples and stories from all over – if you don't know what Chutzpah is, you soon will from *Tell Me A Story*.

Everyone interested in writing better requirements should at once read this book. Academics will find many stimulating ideas for things that ought to be researched; practitioners will be able to reflect in an easy-to-read book on their daily work; students will get an insight into one of the leading minds in the field of human knowledge.

A Retrospective on Knowledge Acquisition

Knowledge Acquisition for Expert Systems A Practical Handbook

Editor: *Alison Kidd*
Plenum, 1987
ISBN 0306424541 (boards)

Knowledge Acquisition for Expert Systems

Author: *Anna Hart*
Kogan Page, 1987
ISBN 185091091X (boards)

Back in the 1980s, no-one had heard of Requirements Engineering, and indeed when I first did (from a respected co-author of mine ...) I wondered how you could possibly call anything as simple as writing down a requirement 'engineering'. But I digress. Back in the 1980's, when primitive beings with hand-axes, simple earthenware pottery and 'expert systems' ruled the planet, phrases like 'Knowledge Elicitation' and 'Knowledge Acquisition' were common - indeed, they were all the fashion. In those far-off days, hominids with large brains held the belief that knowledge could be extracted from people's heads and bottled up in 'knowledge-based systems' which exhibited 'artificial intelligence'. However quaint (or cringe-making, if you were close enough to it) this superstition may now seem, the fact is that a large number of sophisticated techniques were developed, or more likely purloined from established disciplines such as psychology and human-computer interaction, for finding out about what was inside people's heads. Requirements Elicitation existed before Requirements Engineering!

Since I have already made one admission of my age, let me make another. I wrote a book review of Alison Kidd's excellent *Knowledge Acquisition for Expert Systems* in 1989. For curiosity's sake, here it is. I will then attempt a retrospective review and a brief comparison of Kidd and Hart.

***Computing*, 17 August 1989**

The big danger with books containing chapters by different authors is that they frequently become disjointed, written in a hotchpotch of styles and full of contradictions. [Some things never change!] It is pleasant to find a really well-edited book on a subject which is much in need of a sound scholarly approach, and some practical guidance. [Hmm, somewhat pious tone.] It is comparatively short, with 189 pages including at least a page of references in each of the eight chapters. But for the attributions in the chapter headings, it looks like the work of a single author. Each chapter manages to introduce a practical problem in knowledge acquisition, briefly reviews past work, and describes, authoritatively, the most modern approaches to a solution.

Alison Kidd, apart from placing her editorial stamp on every page, has written a crisp introductory framework to the book. Characteristically, she poses some tough questions, like 'What is the relationship between knowledge and language?' and 'What constitutes a theory of human problem solving?'. Kidd dismisses the Feigenbaum analogy (mining jewels of knowledge out of the experts' heads) as misguided. No such simple correspondence of knowledge to things experts say when interviewed exists. Instead, knowledge acquisition is a process of interpretation: it depends on the knowledge engineer's own model of the domain, of reality. [A precursor to the term 'requirements engineer'...]

Two Dutch authors, Breuker and Wielinga, describe a systematic and principled approach for acquiring knowledge. It is worlds apart from the naively enthusiastic candy-box approach to artificial intelligence. [Mmm, thank goodness, I was a bit sceptical even then.] Their approach has been implemented as an interactive support tool and it is well explained. [Is that all? My original review must have been ruthlessly chopped by the editor.]

Other chapters describe in turn how to make a qualitative structural description of some mechanism, how to extract knowledge from transcripts of interviews -- a dreadful task -- and to model an expert's conceptual structures.

Leslie and Nancy Johnson discuss what they call teachback interviewing. When you can teach a topic back to the expert who just explained it to you, and he [oops!] agrees with your version, then you share the same concept. This is certainly more secure than a simple question and answer interview.

Repertory grids and hierarchies of concepts are simply and clearly explained in the next two chapters [by Mildred Shaw & Brian Gaines, and by John Gammack respectively]. These are useful and powerful techniques, and every knowledge engineer should at least know about them.

Finally, Anna Hart looks at the controversial topic of rule induction. If you discover that most pensioners are bald or have grey hair, can you make a general rule with any predictive value? Hart wittily deflates some of the more outrageous claims [Ah yes, I'm sure I gave more examples here], while discussing the topic in enough detail for the knowledge engineer thinking of using an induction package.

This book is a welcome breath of fresh air in a murky subject. It is suitable as a general introduction for students and would-be researchers, and helpful as background for industrial projects. The only negative aspect of the book is the high price. [It cost $39-50, which must be $70 or more at 2003 prices; and it's still in print at $78-50.]

© Ian Alexander (1989)

Knowledge Acquisition, Requirements Elicitation - A Retrospective

A first step in any backward look is to find out when a movement started.

- G.A.Kelly's *Psychology of Personal Constructs* was published by Norton back in 1955. In other words, psychologists were thinking scientifically about how to elicit beliefs half a century ago, and a quarter of a century before his Repertory Grids were applied as a Knowledge Acquisition technique.
- L.A.Zadeh's work on fuzzy logic - applied in the Artificial Intelligence (AI) movement to induce nearly-true rules from examples - dates back to 1972.
- R.C.Schank and R.Abelson's *Scripts, Plans, Goals and Understanding* was published in 1975 by Erlbaum.
- Peter Checkland's *Systems Thinking, Systems Practice* was published by John Wiley in 1981. His Soft Systems Methodology was applied to help fit techniques such as Kelly's "to the ill-defined problems typical of expert system applications" [Kidd, p133].

E.A.Feigenbaum and P.McCorduck's *The Fifth Generation* (i.e., expert systems) came out in Pan in 1984; the ideas had been around in papers for 5 years or so.

The next step is to ask whether progress has been made: for instance, whether a classic book has actually been superseded.

Here are some of the techniques described in Kidd:

- 3 cycles of knowledge analysis and elicitation - orientation, problem identification, problem analysis (with tool support);
- observation of experts at work; representation of qualitative knowledge about structure and behaviour, by analysis of verbatim transcripts ('protocols') and then domain modelling with a network of quantities related by ordinary or differential equations;
- informally from verbatim transcripts by highlighting interesting points, identifying relationships, and refining into if..then.. rules;
- semistructured interviews based on Pask's conversation theory, analysed with systemic grammar networks;
- Kelly's repertory grids, Checkland's SSM, and Pask again, leading to rules;
- tutorial interviewing, card sorting, multidimensional scaling, repertory grids, matrix techniques, and proximity analysis;
- rule induction (with tool support) validated by independent interview.

The one thing that is noticeably missing from all this is any notion of collaboration between experts, and hence of any techniques such as workshops to elicit process knowledge from several people at once. Apart from that, don't we get the impression that this is a rich and varied set of approaches, probably considerably better than the usual round of interviews and jumping-to-conclusions that we know today? So, although a dozen or so years is rather a short time to identify a classic conclusively, I suggest that Kidd is well worth revisiting.

The aspect of Kidd's book that is as dated as beehive hairdos and 12" black vinyl records is of course the emphasis on expert systems themselves. What makes this an unusually good book on the matter is the careful and skeptical attitude of most of the authors. For example:

> Have we extracted all the knowledge possible from the transcripts? We do not know. ... Although we believe in our judgement, it is completely intuitive and we cannot be dogmatic.
>
> We are even less confident about knowledge that may be implicit... "Commonsense" knowledge, general problem-solving strategies, "deep" knowledge of the .. foundations of the domain .. are likely examples of the sort of knowledge that a human expert may not think to mention. [Fox et al, p85]

Tacit Knowledge is perhaps the key issue that made the AI enterprise unworkable; the (unstated!) assumption that expertise could be elicited by interview and captured into hard-and-fast rules was simply wrong. So it is interesting to see that by 1986-7 the foundations were already cracking.

> However, what Feigenbaum terms *knowledge engineering*, the reduction of a large body of knowledge to a precise set of facts and rules, has already become a major bottleneck impeding the application of ESs [expert systems] in new domains. We need to understand more about the nature of expertise in itself and to be able to apply this knowledge to the elicitation of expertise in specific domains. [Shaw and Gaines, p109]

In other words, it was already proving unexpectedly slow and difficult to acquire, extract, elicit, codify the knowledge from experts' heads into rules. Somehow, it just wasn't working too well.

> The methodology described is very easy to implement and run. Experts enjoy using the system and need far less attention from a knowledge engineer. .. It is essential that the expert become familiar with the [approach] .. by experience ..
>
> we put great emphasis on the need to be able to validate any technique for knowledge engineering. It is not sufficient to show that reasonable expert systems can be developed. One must attempt to evaluate the accuracy and

completeness of the knowledge transfer. This is not an easy task ... [Shaw and Gaines, p130]

This is fascinating. The authors are one moment full of childish enthusiasm for their new gadget; the next, they accidentally admit that experts themselves need to become familiar with the elicitation techniques by non-verbal means! Then they pour cold water all over the enthusiasm of others by insisting, quite rightly, on systematic validation -- which turns out to be just as impossible as elicitation.

Induction is consistent and unbiased, although it probably uses only one form of reasoning. Rules are relatively easy to understand... [Hart, p187]

Rules are nice if you can get them. If you can prime your induction engine with a good set of examples covering all eventualities, the engine is guaranteed to come up with good rules. Otherwise, alas, it is guaranteed to come up with garbage, making over-generalised assumptions and risky extrapolations from limited data. To her credit, Hart does say that they must be validated "preferably with both documented examples and the expert."

So, in one way Kidd documents one of the great failures of software engineering, but in another she puts together a powerful set of techniques for getting the most from interviews.

The fact that Anna Hart's book had the same title shows that the subject was immensely fashionable, at least as hot as all things Object and UML today. Her book is less interesting to us now; it has a narrower range and a less reflective outlook. Its heart is a tutorial on rule induction, though there's a good and helpful chapter on Repertory Grids (easier than Kidd's) and introductions to reasoning and probability theory as well as a simple discussion of interviewing. There's also a chapter on the qualities required of a knowledge (aka requirements) engineer, including 'empathy and patience', 'persistence', 'tact and diplomacy', 'logicality', and 'domain knowledge' - though Hart admits this last is "not essential". Apart from the sprinkling of AI terms, the chapter would do fine today as an introduction to RE.

Finally, do I stand by the review written by my earlier self? Well, yes.

Engineering Distributed Objects

Wolfgang Emmerich
John Wiley 2000
ISBN 0471986577 (boards)

Michael Jackson saw me hesitating about whether to read this book and said in his characteristic way:

"It's the Silkworm Principle."

Of course he was delighted to get a bite. "Silkworm, Michael?"

"Silkworms only live in Mulberry trees, and only eat Mulberry leaves."

Hmm, I thought, trying rapidly to reason *au façon* Jackson. If silkworm(X) and eats(X, Y) then mulberry(Y). That makes me a Silkworm. Oh dear.

"No, it doesn't look like Mulberry," said the Silkworm, "but I'll take it anyway."

Teaching point made. Of course, it's impossible to read all about RE let alone all about everything else; but one advantage of not really being a silkworm is not being rule-bound. Wolfgang Emmerich's purpose in this finely-crafted book certainly isn't RE, but there is in fact an excellent section in the review chapter on Distributed System Requirements. It is certainly the best treatment of that specialised topic I've ever seen: a taste of Mulberry at least.

The rest of the book is an extraordinarily clear and readable account of the challenge of creating clean and 'transparent' distributed systems, built up from canonical software Objects as far as possible, and illustrated in UML and C++, Java and CORBA.

Non-Object-Oriented caterpillars will have to make do by skimming the text to locate the introductory sections, the pedagogically clear and accurate diagrams, and the key points (which helpfully have little key icons beside them).

The really serious and difficult requirements for distributed systems are not functions but -ilities (or qualities): portability, scalability, and reliability (including fault tolerance).

Another crucial quality is the transparency already mentioned: the ability of a complex heterogeneous system to appear to users as if it was 'a single integrated computing facility'. Because systems are complicated, it takes careful and systematic design to make all the machinery in the middle - between you and the data you want, for instance - invisible or transparent. The components that Emmerich describes indeed form a new layer or stratum of system structure, 'middleware', between shared services and individual clients. Transparency is not one but many requirements: performance transparency, concurrency transparency, location transparency, migration transparency to name just some of them. In fact you could argue that transparency was the key meta-requirement for distributed systems; it governs all other requirements and design approaches, but is itself more of a far-distant goal than a simply-definable requirement.

Constraints such as heterogeneity are also extremely important in distributed systems. It is so likely as to be almost certain that any large system will encompass diverse hardware and software components, many of which may represent substantial earlier investments. The distributed system builder must live within these constraints.

This looks like an excellent, clear, and comprehensive book that brings the growing and important topic of engineering large distributed software systems to a wider industrial audience. There is a helpful index and a detailed bibliography. Students will find that Emmerich's teaching skill is as great as his technical expertise. Requirements caterpillars who want to keep up with the times would also do well to look at it.

The Domain Theory

Patterns for Knowledge and Software Reuse

Alistair Sutcliffe
Lawrence Erlbaum Associates 2002
ISBN 0805839518 (boards).

This is a remarkable and daunting book (starting with the title), and it is a little difficult to know how to approach it. As if realizing this, Sutcliffe begins the preface not with a description of the intended audience or the book's message, but with:

> If you have picked this book up and want to know what it is about, the answer is software reuse with a difference.

If we add that the book is also about (in no particular order) scenarios, requirements, object modelling, analogy and the basic mechanisms of human thought, something quite new called The Domain Theory and Object System Models (that's chapter 4), not to mention generic tasks and metadomains (that's chapter 5) and lots of deep insights into what knowledge is and how reuse can be based on tasks, claims and models, and the simple fact that much of it applies to systems of all types (not only software), then you'll either become confused or seriously excited.

This is indeed a book that sets out to rewrite a whole field of research and practice. The ideas are based on Sutcliffe's long-standing collaborations with other researchers, notably Neil Maiden (and other CREWS people), Jack Carroll and numerous workers in the realms of HCI and patterns, including Jim Coplien who wrote the Foreword.

What, then, is the Domain Theory itself?

> From the outset the Domain Theory had an ambitious claim: that all software engineering problems could be described by a tractably small set of models that reprsented fundamental abstractions. The focus .. was requirements engineering..

In other words, many people are working on reusing design components such as software objects; but at deeper levels than things and design patterns are people's goals, tasks and reasons for doing things. Reusing these would indeed be powerful.

But isn't Michael Jackson's work on Problem Frames precisely about reusing requirements patterns?

> Jackson's problem frames and domains are more abstract than the Domain Theory, which models generic knowledge closer to the real world and with more detail; however [it] does share Jackson's concern about defining the focus between the designed system and the external world.

Sutcliffe thus claims simultaneously to be powerfully generic (and hence abstract) and directly practical, giving enough detail for day-to-day reuse on real projects. This is startling stuff. Ultimately, any proposal for reuse succeeds or fails on the quality of the things that it suggests can be reused. A hundred pages of the book are therefore rightly devoted to appendices. These list in turn

- the specifications of the Object System Model (OSM) library, which includes things like a model of Object Servicing and Repair; this leads to neatly-reusable generic requirements for the various tasks involved.
- generic Task Models, Dialogues, and Argumentation Schemas (shouldn't that be schemata?), which includes things like a goal hierarchy for the diagnosis task (of which repair is a subtask); and finally
- a Claims Library, which includes things like an Event Filters Claim, which argues the toss over the 'upsides' and 'downsides' of a gadget to filter out unimportant sensor reports as variables like temperature and humidity wobble about - letting key changes like the sudden rise in temperature and smoke in a fire through.

It is difficult to judge the importance or practical impact of a book as radical as this. I suspect that Sutcliffe and his many more-or-less unsung co-workers are on to something major here - it is hard to imagine that a mere library of lumps of software will ever be sufficient in itself. On the other hand, it is rare for a book to make a serious impression on a world that is not especially literate or studious: Christopher Alexander's effect on the Gang of Four (another precursor of the Domain Theory) is a rare and perhaps fluky exception.

Jim Coplien's claim is however probably correct:

> The foundations in this book are a treasure trove for contemporary design. ... I am confident that any student of design will find strong links between their research and experiences in the technical domain and the deep insights that Sutcliffe offers into who we are and how we work.

... but only if they read this book. If you want to be one of the lucky ones, get a copy now.

RE-Sponses

During the "Scenarios Work" event held at UCL in July, Roland Leibundgut from Zuhlke Engineering hinted at a very interesting aspect of RE textually written Use Case description which was reported in RQ-27 as, "*A Use Case was not just a bubble on a diagram but could be as much as 80 pages of text; more than 3 pages was common*".

The point to emphasise here is that Roland felt, quite legitimately in my view, the need to give the audience a quantitative evaluation of the requirement work he was talking about. I found this need refreshing as most of the RE specialists focus solely on the qualitative aspects of the requirements description. Therefore we describe the functionality of requirements very well but are often unable to evaluate how much functionality we have in this requirement. Even though Roland's point was about the metrication of the requirement work, the pages-of-text is also still used by many organisations in sizing up the functionality of their requirements. This concern is significant because if we want to select a requirement of interest among competitive ones with a view to developing the particular one, project management finds it important to be able to size up the resulting software very early in order to estimate and plan the development project.

In the context of the functional sizing of a Use Case, there exist modern measurement standards such as COSMIC-FFP. Already in use for several years, recently promoted ISO standard, COSMIC-FFP has a freely available documented method (http://www.lrgl.uqam.ca); in addition to this, training course (http://www.sms.uk.co/) and support tools (http://www.telmaco.co.uk) exist.

Recognising that, using a work metrics, Roland did not intend to specifically express a functional measurement of his Use Cases, I would still like to recommend a closer relationship between qualitative and quantitative descriptions of the Functional User Requirements.

Bernard Londeix, Telmaco Ltd, 26/12/02

RE-Sources

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

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

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

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

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

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

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

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

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

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

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

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

Mailing lists

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

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

subscribe RE-online <your email address>

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

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

RE-Actors

The committee of RESG

RE-Creations

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

Deadline for next issue: 1st May 2003.