Requirenautics Quarterly

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

http://www.cs.york.ac.uk/bcs/resg/

 Issue 20 (March 2000)

RE-Locations

RE-Soundings

Editorial

Welcome to RQ20. It's something of a bumper issue this time. The tireless Ian Alexander has written two reports on recent RESG events and reviews of two recent books on RE.

In the papers section, there's an industrial experience report by Godfrey Draper and Peter Robinson about requirements management in a large telecommunications project, and a position paper by Dan Berry on human, methodological and managerial difficulties with problem understanding. One of Dan's points is that methods and tools can't 'do' problem understanding, but if they help people co-operate, then they might ease the understanding process.

This is echoed in this issue's Core-blimey where Geoff Mullery proposes just such a tool. Geoff's vision is (I paraphrase) of an Internet that's useful. Read his ideas and you'll see that what he proposes is (probably) technically feasible with recent developments in authentication mechanisms and data mining (among others). I know Geoff is keen to stimulate debate on this. Perhaps, over the next few issues, we could develop a conceptual model of a tool that meets Geoff and Dan's requirements. Ideas, requirements, constraints and dissenting views most welcome.

As always, opinions about, and suggestions for RQ's structure and content are also welcome. Even better, send us a contribution for publication. We're particularly keen on reports of experience, position papers and dissent from the orthodoxy. Right. Time to clear the bustling RQ

production office for issue 21. I'd better just move the editorial bike out of the way…

Pete Sawyer,
Computing Department, Lancaster University

Chairman's Message

I am increasingly attending meetings labelled as RE events, where participants present and discuss work bridging or applying RE techniques to other areas of software and systems engineering. I find this encouraging because I have always believed RE to be a multi-disciplinary activity, with applications across a wide range of systems development situations. In February, I attended an IFIP Working Group 2.9 meeting on requirements engineering, where the complex relationships between requirements and design were discussed. Also in February, the RESG organised a meeting on "Dependable Distributed System Requirements", reported in this issue of RQ, where the intersections of RE and Distributed Systems development were discussed. In March, the RESG and the Advanced Programming Group teamed up to debate and discuss the relationship between RE and Extreme Programming.

I find such meetings encouraging because of the recognition they implicitly give to the fact that requirements engineering has an impact on many different aspects and stages of the development life cycle.

In May, Jim Arlow takes some of these issues forward by discussing, in a hands-on mini-tutorial organised by the RESG, how the ubiquitous UML deals with requirements. Wouldn't you like to know how UML does this?! Well then, register early to guarantee a place at this event! (attendance is limited to 30 delegates).

Finally, a note on RESG membership. You will soon receive, if you haven't already, renewal notices asking you if you wish to renew your ERSG membership. Once again, membership of the RESG will be for two years, to minimise the administration on you - RESG members - and Membership Secretary, David Shearer. This is David's first go at membership renewals since taking over from Sara Jones last year, so I hope you will assist him in returning your membership renewals as soon as possible to ensure your delivery if RQ is uninterrupted. I hope you feel that membership of the RESG has been good value for money, and I hope that you feel that it is worth continuing your membership. As always, I emphasise that the success of the group is due to the participation of you, the members, and the huge voluntary efforts of the committee officers. Thank you all for your support!

Bashar Nuseibeh,
Imperial College, London

Industrial Liaison - let us know when you change your email address

In these days when change is the norm, one thing is almost certain: sooner or later you will change your e-mail address. When you do, please let us know so we can continue to keep you right up to date on RESG activities and other events of special interest to industrial members. The person to tell is David Shearer d.w.shearer@herts.ac.uk, with copies to myself at b.farbey@cs.ucl.ac.uk and Carol Britton at c.britton@herts.ac.uk.

Barbara Farbey
University College, London

RE-Treats

Next event organised by the group.

Using UML in anger for Requirements Engineering - a mini tutorial

Presented by: Jim Arlow
Date: 17th May 2000
Location: Imperial College London
Contact: Bashar Nuseibeh, Imperial College, London (ban@doc.ic.ac.uk)

Related event of interest to RESG members:

Mastering the Requirements Process

Presented by: Suzanne and James Robertson
Date: 27th-29th March and 18th-20th September 2000
Location: London
Contact: IRM UK (http://www.irmuk.co.uk)

Note: RESG members are entitled to a 10% discount.

Other forthcoming events.

Scenarios (to be confirmed)

Date: 17th July 2000
Location: London
Contact: Ian Alexander, Independent Consultant (iany@easynet.co.uk) and David Shearer, University of Hertfordshire (d.w.shearer@herts.ac.uk)

Requirements for 1 Billion Users - RE for websites and product lines (to be confirmed)

Date: 29th November 2000
Location: London
Contact: David Shearer, University of Hertfordshire (d.w.shearer@herts.ac.uk) and Barbara Farbey, UCL

(b.farbey@cs.ucl.ac.uk)

RE-Calls

Recent Calls for Papers and Participation

ICSE 2000 – The New Millennium: 22nd International Conference on Software Engineering, Limerick, Ireland, June 4-11, 2000
http://www.ul.ie/~icse2000

CAiSE 2000 – 12th Conference on Advanced Information Systems Engineering, Stockholm, Sweden, June 7-9, 2000
http://www.ul.ie/~icse2000

REFSQ 2000 – Sixth International Workshop on Requirements engineering: Foundation for Software Quality, Stockholm, Sweden, June 5-6, 2000 (Preceeding CaiSE 2000)
http://www.ifi.uib.no/konf/refsq2000/cfp2000.html

ICRE 2000 - Fourth International Conference on Requirements Engineering, Schaumburg, Illinois, USA, Limerick, Ireland, June 19-23, 2000
http://www.cse.msu.edu/ICRE2000/

REP 2000 - Second International Workshop on the Requirements Engineering Process, Greenwich, United Kingdom, September 6-8, 2000
http://panoramix.univ-paris1.fr/CRINFO/REP/

RE-Readings

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

RESG Meeting on Requirements and Commercial-Off-The Shelf (COTS) Systems

This out-of-town meeting was held at the University of York on the 23rd November 1999. **Laurence Brooks** welcomed about 25 of us including a London contingent to what he called the rugged mountainous North, which turned out to be a quite hospitable place with tea and cakes.

Phil Osprey, IT architect at Marks and Spencer, spoke on Developing a Business Driven IT Strategy and Issues with Software Components In Retail. His job was to develop a business-driven IT strategy, and to deal with software component issues in Retailing. "VISIO is one of our main tools", he said, going on "we are a retailer not a software house". The recent re-organisation of M&S had been made without evaluating the impact on business processes, so the IT department was involved rather late.

Retail functions had been matched quite closely to business goals, but the older systems could not be extended and the old ICL hardware they ran on was no longer manufactured. Rapid change was needed – in 3 months.

Analysis was simple and pragmatic, using traditional flow diagrams and transaction rates. None of the well-known ERP offerings met M&S's needs; the chosen solution demanded that 36 applications be integrated. Osprey was tempted to "bend the process" to fit into an ERP tool, but the ERP supplier was not keen to pay for changes unless these were generic.

Was the M&S approach special? It had been late into IT because the manual system which lasted into the 1970's was very efficiently documented in "The Book". It had counted stock and analysed takings from tills, in a mechanism which was costly, good, but idiosyncratic. For instance, it derived sales figures from stock changes!

In the end, COTS packages from a dozen suppliers were integrated; about half the functions consisted of bespoke code, and some of the COTS functions were up to 50% bespoke as well.

As for the platform, Microsoft was the only option for in-store use as 90% of the packages ran only on that. Mainframes could offer at most 30 sources of software. Migration continues; in 2001/2 customers will be able to order anything by Internet and collect it by car from their local branch.

COTS offered retailers lower cost, lower risk, shorter time to market, and greater reliability. Against those advantages, there were both too many standards and a lack of standardisation; there was self-interest to look out for, and no obvious brand leader to choose. Osprey expected a shake-out among software suppliers, and thought that XML and a shared data model might tidy up the interfaces.

Ljerka Beus-Dukic, University of Northumbria, spoke on The requirements for a COTS software component: a case study. She was working on a project called GUARDS:

Generic Upgradable Architecture for Real-time Dependable Systems. A COTS package was a piece of software from a commercial vendor, performing complex functions for many customers in the form of a "black box" – rarely with source code. The idea was to find something small enough to deploy, but big enough to support effectively. This brought advantages through being generic, composable and reusable.

In the case of real-time systems, this meant a processor and a backplane bus in hardware, and an operating system in software. The problem could be characterized in three dimensions: integrity level, number of channels (for n-redundant fault tolerance) and intra-channel multiplicity. Standard Requirements Engineering processes, methods, and techniques such as elicitation, interviewing, scenarios, prototyping, and dataflow models did not seem appropriate.

But requirements were needed. The architecture had to be transparent, scaleable, and able to support distributed real-time operations. Software applications had to be segregated, reliable, and have a long deployed lifetime. Development projects had to have a proper environment and support from a credible supplier (a business requirement). And critically, there had to be compliance to 3 specific strands of the POSIX standard: 1003.1a, b, and c. "This was not easy to conform to" remarked Ljerka drily. What was true conformance? How could vendors be compared fairly? How much domain knowledge did you need to make a choice? What about politics, legacy systems, and budget?

In the event, two of the project partners chose Oses, which the project checked very carefully. The documentation did not substantiate all the claims made even by reputable vendors: for instance, some OS calls were just empty shells!

The lessons were that requirements had to be in sufficient detail, as did the information from COTS vendors. It was nice to have independent facts from the POSIX standard itself. There should be requirements on legacy and future implementations, to include the framework as well as system components. And it was essential to be able to retrieve requirements efficiently. Users liked to follow other users' choices: in practice, word of mouth was both safest and easiest.

James Devlin, City of York Council, spoke on Industrial Experiences of Requirements and COTS. He agreed with many points made by the earlier speakers. In his case, choice of suppliers depended on the product: there was just one supplier for air pollution monitoring, but 100 of them competed for time recording. It was dangerous to follow user preferences for suppliers: his users generally did not have personal experience of the software. For example, there had been a painful experience with an ODBC database recommended by someone: to get a report out of the database, he had had to commission an export script to create an Excel file so users could write a report from that.

Careful evaluation through interview proformas, vendor presentations and site visits, and prioritisation of requirements into essential, highly desirable, desirable, and wishlist, helped weed out the vendors and get the products that were genuinely needed.

Godfrey Draper and **Peter Robinson**, Computer Sciences Corporation, spoke on Requirements, COTS and Telecomms. Their client was ICO, a mobile telecomms utility which intends to start trading soon. ICO plans a network of 12 satellites with ground stations, so it will have a large capacity. Standard functions such as global positioning are handled with COTS products, interfaced as usual by bespoke code. COTS products formed a large percentage of the £100M cost of the system. CSC, acting as system integrators, had 100 engineers on the project. In something of that size, it was vital to have a requirements trace back to the ICO contract.

The requirements were managed in ISDE's Tracer tool using a single licence; engineers were given supposedly read-only copies on Excel spreadsheets. One engineer had the bright idea of importing the spreadsheet into an Access database and tracing the requirements from there, leading to some confusion, so the economical strategy was not without risk. CSC used an attribute for "CSC Comments" to qualify the ICO requirements, and by agreement the contract.

80% of the requirements were straightforward business ("user") requirements; 10% were on the development process, and 10% were technical. Of this initial set, about 15% had been "retired" by the time of the first release.

What CSC would have most liked to do was to be able to say "all these have been verified" in the case of many-to-many traces between requirements and design elements and tests. The tool would have had to chase (recursively) down all the traces, find out if each one led to a successful verification, and come back with a simple result for each requirement. The better tools like DOORS and CALIBER can do this. However the simple traces offered by Tracer were vital for control. Requirements showed when the customer had to pay, made reporting easy, enabled control of development, reduced ambiguity, and allowed impact to be analysed for maintenance changes. *(a paper describing this work appears in the next section - ed*)

Douglas Kunda, University of York, spoke on the STACE Framework, a Socio-Technical Approach to COTS software Evaluation and selection. The problem of choosing among COTS products consisted of limited documentation, blackbox products, rapid market changes, and poorly defined selection processes.

The STACE solution was cheap and systematic. It used participative design and analytical-hierarchical process (AHP) decision theory. And it took social and economic factors into account. It formed a simple framework for evaluating tenders from different suppliers.

The method was to define the problem with requirements, and then prepare for assessment by defining criteria to distinguish between better and worse products. Candidate products were then identified and characterized. Finally, the rival products were assessed. The calculation derives an eigenvector from a matrix of the relative importance of the different attributes such as functionality, quality, and socio-economic factors. For example, if function A is as important as function B, you enter a 1 in the matrix, whereas if A is far more important than B, you enter a 9 in the matrix. The matrix is squared repeatedly until it stabilises. The resulting eigenvector gives the weights or "priority rankings" of the criteria. This is then multiplied by each product's set of raw scores on the criteria to give a single score for each product.

RESG Half Day Seminar on Dependable Distributed Systems Requirements

The RESG Half Day Seminar on Dependable Distributed Systems Requirements was held on 16 February 2000 in University College London.

Wolfgang Emmerich (UCL) welcomed everyone to the meeting. He felt that distributed system architecture was driven by constraints (non-functional requirements). In particular, the bundle of constraints known as Dependability – safety, liveness, stability, security – often made systems what they were.

An obvious issue for the RESG was the connection between requirements and architecture. There was a dogma in RE circles that requirements should be free of design constraints. The meeting would look at this from various perspectives.

David Bush (National Air Traffic Service) asked 'Do Distributed Systems Need a Particular Approach to Requirements Engineering?'.

NATS turned over £550M per year with a staff of 5400. It handled 1.8M flights, up 7%, its charges were down 5%, and it received no UK government funding.

It made use of safety related standards (ISO) 61508 and 178B but neither were 100% relevant. Key issues in the business were traceability, completeness, and good practice, all to do with RE.

NATS sought advice on RE tools from UCL, but soon realized that there were differences between projects and between business areas, and that varied tools were in use. A major problem was the client/supplier contractual divide. There was no pan-European philosophy on RE but NATS urgently needed a standardized information model, process, and set of tools. The information model had to be goal directed, sensitive to Jackson's Machine/World distinction, and include architectural design in the process. Goals helped NATS to understand if a model was complete; the world and machine focus forced people to think about system boundaries.

"Aircraft are not inside our boundary. You may then ask why we call it air traffic ***control***", said Bush.

Distribution was often hidden from users, so it was not a user need but a design issue. Maybe it was better to think about stakeholders and goals than users and requirements, he said. The audience stroked its collective chin thoughtfully. NATS had a system that had evolved to become complex, with much equipment not under its control. A component-based approach was forced on it by distribution.

Jeff Magee (Imperial College London) spoke on Architecture, Analysis & Animation. He worked with Nat Pryce, Jeff Kramer, and Dimitra Giannakopolou on graphic animation of behavioural models and animating scenarios. Analysis was by model checking.

The traditional (well, quite new actually [Editor's note]) approach to RE was to model stakeholders' goals, use cases, assumptions, constraints, and properties. Then one went on to select an architecture, make models, do analysis. What was missing was feedback to validate all of this with the stakeholders.

"Graphical animation closes the loop" said Magee.

Animation can check scenarios, help stakeholders to join in, show safety or progress violations visually, and even help with model building.

Formally, the group was using Label Transition Systems, which looked pretty much like State Transition Diagrams to you or me. They drew 2-D graphics – a gantry picked up a component, carried it over to the drill bench, then to the oven, then to the conveyor belt. If job A needed the drill then the oven, while job B needed the oven then the drill, they obviously got stuck when they each hogged the other's next resource.

The LTS model checked progress properties such as deadlock and reachability, and pointed out exactly where things failed in the users' language. You could build up animations by composing actions into sequences and parallels. They used Java and XML "Scene Beans" to go from graphics primitives like line and circle, to behaviours like rotate and translate, to animation threads. Animation movement represented logical clock time so users could see for themselves what was being proposed.

The approach was applicable to a wide range of systems, not necessarily at all like robotics.

Anthony Finkelstein (University College London) spoke on Architectural Stability, or at least the problem he had with it.

Software architecture was a structural description of assemblages of components and connectors. There were

perhaps rather few choices – client/server, pipeline, layered, pipe and filter, etc.

Requirements set real-world goals for services provided by architectural structures, and also set constraints on them. Therefore requirements and architectures intertwined tightly: it was hard to imagine (or specify) a system without an architecture; and there was an interplay of the achievable and the desirable (from the start).

Most development was Brown Field – the existing architectural context is important in Requirements Engineering, despite all the propaganda. In component-based construction, requirements might seem to be all that mattered, as the design was already inside the components. In reality, requirements, cost models, components, and architecture must be co-developed.

Requirement change is a good thing; as Manny Lehman's laws say, systems change the world and create the need for further change. In the face of this, architectures must provide a haven of stability – for the cost of restructuring can be forbiddingly high.

The problem, said Finkelstein, is that some requirement changes "break the architecture". Could we find a way to measure and predict this?

Most of the ill-behaved changes were to constraints, global system properties such as performance, scalability, and security. If you ask a small PC-based web server to scale up or go faster, you can't just buy a bigger CPU, you need a 3-tier architecture with queuing systems and so on.

Not all changes are equally likely, so it may be possible to guess what will change. And some architectures allow some kinds of change – one approach may be scaleable but not allow performance gains, or vice versa. If a bank chooses a multi-server approach it may sacrifice security for performance.

Finkelstein closed by setting an agenda of related research problems: can we pick an architectural style to handle expected changes easily? Can we assess the impact of a proposed change? How do architectures fail – what is brittleness in software, and can we promote graceful degradation? Can we describe these things in a goal model? Can we use an architectural design language to help? Can control theory predict what will be stable?

Jonathan Moffett (University of York) spoke on Security Policies and Requirements in Distributed Systems.

He said there was a Bermuda Triangle between requirements, security policies, and distributed systems. A requirement was "something called for or demanded, a condition which must be complied with". A policy was "a plan of action".

Security policies could be mandatory, hard-coded into systems as if they were requirements, or discretionary, able to be tweaked by administrators on-line.

In practice, risk analysis (as for safety) was separate from system development – the problems are sorted out once systems go into service. Security requirements are often not fully thought out: instead, they are just parachuted in with "comply with Orange Book". This may constitute efficient reuse – it has advantages, certainly – but when you get a ten-page list of relevant standards, it isn't easy to deal with.

Distributed systems with mutually autonomous agents create the need to agree protocols, cryptography, and so on to permit co-operation at any level. The basic philosophies are prevention (access control) and confoundment (cryptography). In real distributed systems, control can be exercised at source, on the network, and on the target: multiple firewalls in the modern jargon.

Panel

There was a short panel session, where the speakers were asked whether distributed systems needed special Requirements Engineering treatment. Moffett said yes, more autonomous agents were needed. Magee said yes, one had to think about partial failures. Finkelstein said no, all systems were distributed! Bush said no, all systems needed better thought-out RE. Ray Offen (visiting UCL) said no, but you had more work to build the domain models. Bashar Nuseibih said that the -ilities (constraints) were the issue. Ian Alexander said that viewpoints were harder to capture in autonomous groups.

It was interesting for those of us who have been working on animation of requirements to see that the issue is very much a live one, far beyond the normal reaches of Requirements Engineering – and how elegant the research approaches are. The thing that needs to go is the belief that requirements and design are separate: no-one starts with a clean slate when specifying and designing a new system.

RE-Papers

Requirements, COTS and Telecomms

By Dr Godfrey Draper and Peter Robinson
CSC Computer Sciences Limited

Editor's note: This paper was presented at the RESG event on Requirements and Commercial-Off-The Shelf (COTS) Systems that took place at York last November (reviewed above)

1 Introduction

This paper describes how requirements management is proving successful in supporting a large commercial System Integration project.

ICO BOSS provides the business support for a global mobile phone system. It is based primarily on COTS packages with inter-linking bespoke development in C++.

The paper describes the requirement database structure, the costs and benefits of managing requirements and the extent to which the theoretical benefits have been achieved in practice. It demonstrates the crucial importance of traceability as an ingredient of success.

2 Background to Project

ICO is a start up company, funded by several major telecommunications companies, which is in the process of creating a new global mobile phone business. This will use a configuration of up to 12 satellites to communicate with handsets. Satellite Access Nodes (SANs) collect calls from the satellites and a fibre optic network to transmit calls around the world, both to other ground based and GSM networks, and also back via the satellite system to other ICO handset users.

The BOSS project is being undertaken by CSC as ICO's systems integration partner for the project. The programme contract is a System Integration project, to be followed by ongoing operations and maintenance. BOSS is designed to provide the customer billing capability for ICO's Business Operations Support Systems (BOSS) and is divided into four scheduled releases. BOSS includes all the enabling technology required to run the business side of a telecoms company. It is a critical element in ICO's planned service launch, which is based on a new generation of satellite-enabled, pocket-sized mobile phones with a telecoms infrastructure that will allow users to make and receive calls anywhere in the world. The project utilises commercial off-the-shelf software (COTS) and bespoke development based on a highly robust IT infrastructure.

The aim is to ensure that, as far as possible, BOSS is future proof, secure, robust and flexible. CSC is using intellectual rigour to insulate ICO from changes in the underlying packages, but primarily to deliver the scalability and flexibility that ICO will need to support its business in a highly dynamic marketplace.

3 Business Operations Support System (BOSS) overview

BOSS:

- collects call data from the switches;
- uses geographical positioning to find position (country and area [territory & admin district]);
- uses rating algorithms to calculate charges;
- enables distribution to Retail and Distribution Partners (RP,DP);
- supports wholesale invoicing;
- provides Partner care to answer Distribution Partner queries.

The BOSS project comprises a number of phased releases. Each release provides added functionality in an incremental manner. The development has been planned to achieve completion in a period between 2 – 3 years, involving well in excess of 100 project staff at peak. It is designed to satisfy around 1600 requirements, divided into three categories of business, architectural and technical requirements.

The majority of the functionality is provided by a number of COTS products, with bespoke development deployed where a suitable alternative is not available. The size of the project and the variety of components deployed in the solution adopted, necessitate the ability to manage requirements effectively.

4 Requirements Trace Matrix

Requirements Management is the cornerstone of the contract for the development and integration of software and COTS products. Requirements were listed in the contract, with a defined set of fields, primarily :

- Unique ID : A numerical identity for each requirement for reference purposes.
- Source document name (ARD, BRD, TRD): The requirements were initially defined in 3 source documents. ARD defines the Architectural requirements, how the system is to be structured, including maintenance and performance requirements. BRD defines the Business requirements, what is to be done to support the business. TRD defines the Technical requirements, mostly the processes and methodology to be used in development, including Configuration Management and Quality Assurance.
- Reference paragraph: the paragraph number of the ARD, BRD or TRD containing the requirement.
- Requirement text: self explanatory.
- CSC comment: this property was part of the contract and qualified the requirement text. Examples are comments like:
  - *Manual process in release A.*
  - *Error messages will be stored in a log file accessible to all users.*
  - *Data can be exported to a spreadsheet for generation of graphs.*
- Stage: the stage number in which the requirement is planned to be satisfied.
- Release: the release in which the requirement is planned to be satisfied.
- Application compliance: The COTS product that fulfils the requirement.
- Change Order number : The change order pertaining to a change in the requirement (added later).

These requirements and attributes are only changeable by formally agreed contract Change Request, in order to control scope creep and ensure that cost and schedule impacts are understood and agreed. Thus the programme

has kept to schedule and within budget. It also ensures all stakeholders in the project are aware of the up to date requirements definition.

The implementation is planned to be in 2 stages with 4 development and 3 operational releases. Stage 1 provides the minimum necessary for an initial operational system. The releases will be coherent and consist of a set of COTS packages linked by bespoke software. This process of incremental development is designed to manage risk and phase payments. This places particular importance on Requirements Management in order to define the scope of each release.

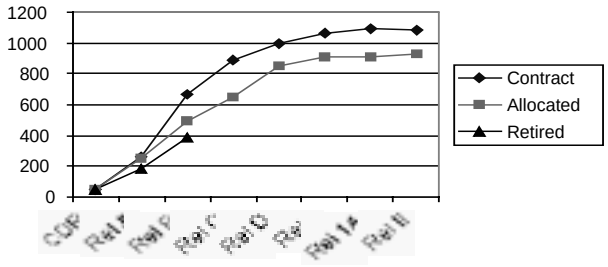

Figure 1 Requirements status

Figure 1 shows the current status of the requirements. The x-axis shows the releases including the Critical Design Review (CDR). The y-axis shows the number of requirements initially contracted to be satisfied by the release, the number allocated to each release following change request deletions and the number retired at review or release. A retired requirement is one that has been demonstrated to have been satisfied either by review or by an acceptance test.

A key reporting aspect is that the requirements are being implemented in 4 development releases, and 3 releases in the operational environment. These need to be assigned, tracked, changed and retired.

There are also several users of requirements status reports during the development life cycle:

- Management (Programme Director, Requirements Manager, Release Managers) to plan the staff and schedule based on workload.
- Business Architects (Solution Engineering) to define the processes to meet methodology requirements.
- System Architects (Design) to establish the design to meet system requirements.
- Development (Development) to develop the detailed design and code.
- Testers (Test and Integration) to define the tests required to verify the requirements.
- Verification of process requirements (Test and Integration).
- Programme release managers (Release Managers) to complete acceptance of each release.

The various stakeholders and the organisational relationship between them are shown in figure 2.

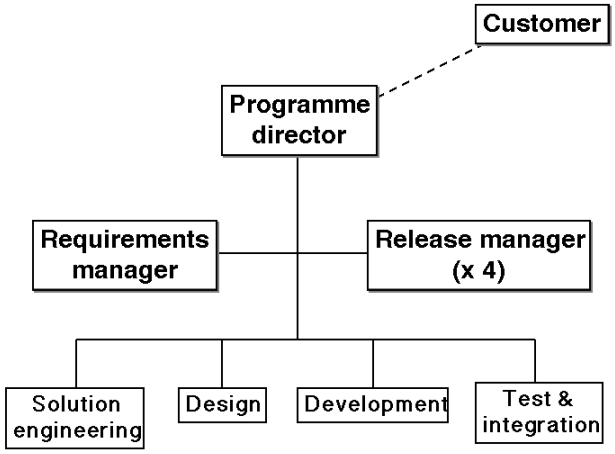

Figure 2 Programme team structure

The use of the requirements evolves through the life cycle according to the roles of the users of requirements status reports outlined above. To manage this CSC uses the internal proprietary methodology called Catalyst ™.

4 Requirement trace database design

The Requirements trace database design was formulated to support the following trace Processes:

Requirements creation, Business Area Architecture (BAA) Reports, Test case/script mapping, Test status mapping, Verification mapping, Requirements retirement, Change Request implementation, Acceptance Authority, Application mapping, Code Object mapping and Scenario mapping.

The requirements trace tool used is a product called Tracer (V3.0) from ISDE Ltd. It consists of a database repository and a set of tools designed specifically to support requirements traceability. The trace entities to be used are established using a configurator enabling the user to define the trace entities and the relationships between them.

In addition to the requirements, the BOSS traceable entities that have been defined are:

- Solution Design – the Business Application Architecture.
- Acceptance test – formally defined and observed test specifications, case and scripts, with status to verify the requirements for a release.
- Verification Report – report references to verify requirements by review or inspection.
- Application Component – COTS or bespoke software names.
- Code Object – code object or class that implements a bespoke software application component.

The Tracer database has been set up with a configuration containing the Requirement entity as the core entity. The links to other trace entities are in a star shaped design as shown in figure 3, with each major life-cycle element linked directly to the requirements. An alternative is a hierarchy from requirements to solution design to detailed design to applications. In both cases, the test and verification data needs to be linked to requirements directly.

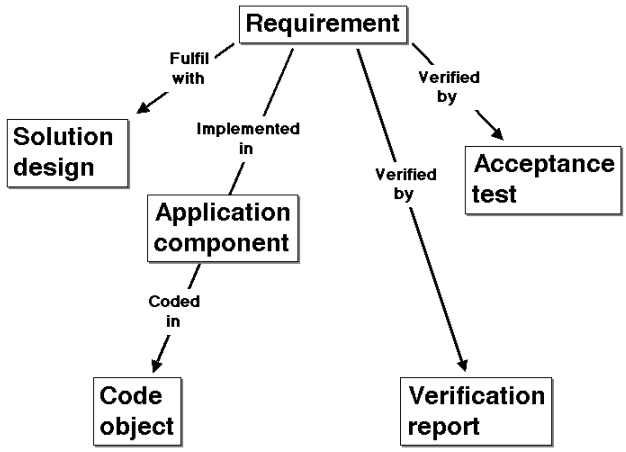

Figure 3 Requirements traceability matrix

In addition to the trace hierarchy configurator, the tool provides a number of user functions which include:

- On line edit;
- Reports;
- Bulk import of requirements data.

The majority of the requirements are implemented by a set of COTS products. These are linked by software developed in-house and by off-shore subcontractors. Each COTS product and bespoke application is linked to the requirements that they implement. This is useful for impact analysis in case of change. When a requirement is to be changed, or the implementation is to be rescheduled, then it is possible to trace to the application that is affected. It is also possible to see how many requirements each COTS product, giving a measure of value, implements, although, of course, requirements do not have equal weight. This is summarised in table 1. The left hand column represents architectural components labeled A to E. The middle 4 columns represent how the requirements are satisfied - by Bespoke software, COTS, a mixture of bespoke and COTS software, or as a human task.

A trace of developed bespoke applications to C++ classes is yet to be implemented. This will provide another level of impact analysis, and also demonstrate reuse of code. It will be particularly useful during the maintenance phase.

Table 1

	Bespoke	COTS	Mixed	Manual	Total
A	23	3	7	21	54
B	9	4	4	0	17
C	4	100	5	0	109
D	0	15	12	1	28
E	14	87	4	0	105
Total	50	209	32	22	313

In order to sign off the development of each release, the requirements are each retired, either by:

- formally observed and signed off execution of a test script that implements a test case; or
- by reference to a document or audit that demonstrates the implementation of the requirement.

The former is most relevant to business requirements, the latter to process or architecture requirements. The Tracer tool provides a user GUI to define the content and structure of reports, which can be produced in Excel CSV or Word format. The user can define :

- the fields of a component to be printed;
- the order in which they occur;
- the selection criteria;
- the sort order;
- additional linked components and their fields.

A simple example is shown in table 2.

5 Management aspects

At contract commencement, an agreed set of requirements provided the starting point or baseline for the development of the system. As the development has progressed, changes have been required for a number of reasons. The main reasons were:

- better understanding of user's needs;
- changes in business objectives;
- phasing of implementation (release definition);
- errors in the original requirements.

A formal change control procedure provides the means to determine and agree changes, which are reflected in the updates to the baselined requirements. Each requirement has been versioned such that a full history of its evolution can be traced, with reasons for the changes recorded.

The phased development through a number of defined releases provided the opportunity to demonstrate satisfaction, or compliance with, requirements in a succession of releases, rather than saving up the 'signing off' of requirements until final acceptance. Such an approval process has facilitated the management of risk to both the customer and contractor. Early demonstration of satisfied requirements becomes feasible and minimises the risk of missing requirements.

The existence and maintenance of a fully populated trace database provided the means for project wide and up to date information, meeting the needs of all the stakeholders identified in section 4 as well as customer representatives.

A wide variety of metrics were maintained including the following:

- Requirements allocated to releases.
- Requirements satisfied by a release.
- Requirements deferred to a different release than planned.
- Requirements for which a design object exist/did not exist.
- % of allocated requirements satisfied.

Trend reports were also produced.

Table 2

6 Summary

Source	Requirement	Requirement Text	CSC Comment	Test Specification	Test Case	Test Script	Test Status
BRD	2.5.1.5.1	2.5.1.5.1. The system shall be capable of carrying out Service Activation at a future specified date, using pre-loaded data.	Agreed				
				15110-TSP-005	aBAT01083	BBAT007	Pass
BRD	2.7.1.3	2.7.1.3. All Service Suspension Requests shall be further validated for the nature of the suspension, fixed (service suspended between specified dates) or indefinite (service suspended until further notice).	Agreed				
				15110-TSP-005	aBAT01083	BBAT007	Pass
BRD	2.9.1.1.3.5	2.9.1.1.3.5. A Service Profile Change Management Request shall contain the following data: Wholesale service package.	Agreed				
				15110-TSP-005	aBAT01059	BBAT008v02m	Pass
				15110-TSP-005	aBAT01060	BBAT016v01m	Fail
BRD	2.9.1.1.3.6	2.9.1.1.3.6. A Service Profile Change Management Request shall contain the following data: Fraud detection profile.	Agreed				
				15110-TSP-005	aBAT01059	BBAT008v02m	Pass
				15110-TSP-005	aBAT01082	BBAT006	Pass

The strengths of the approach were reflected in the effective control offered by baselines and traceability, allowing timely and accurate information for a variety of project needs. Full lifecycle coverage was possible and provided a multiple set of consistent requirements.

The major benefits fell into 4 categories:

1. Customer pays! The process paves the way for phased acceptance and the basis to obtain payment for the work undertaken.
2. Reports and good communications. A wide variety of management/technical reports to meet different information needs. The ability to provide up to date information in a consistent manner, to customer, managers and technical staff.
3. Reduction of ambiguity. The management of the requirements process and the associated documentation provides the means to reduce ambiguity and to maximise the likelihood of acceptance of the delivered system.
4. Impact analysis. The trace information contained in the database will enable the impact of changes to be assessed and the implications studied and costed as a basis for deciding the merits of proposed changes. In particular, it will permit identification of where COTS packages can be deployed to support future change or whether bespoke development is required. This information will enable informed choices to be made and priorities assigned to change requests in the future.

The costs including manpower and tools have been very modest, consisting of a single user license for the trace tool (£3K) and an annual maintenance charge. Manpower effort has included a principal consultant at 4 days per month and a senior consultant for 2–3 days per week, 50 days and 120 days respectively.

The main improvement needed is to ensure that all stakeholders are made aware of the instituted requirements management process at the commencement of the project

and adhere to it. Some attention to the detail of the requirement is also needed to reduce ambiguity.

The timing of changes to requirements is also important and speed of update of the trace database is essential to maintain a consistent view of the current state of development.

Glossary

BOSS	Business Operations Support System
COTS	Commercial Off The Shelf
CSC	Computer Sciences Corporation
GSM	Global System for Mobile Communications
ICO	ICO Global Communications – a company developing a satellite mobile phone system
OOD	Object Oriented Design
RTM	Requirement Trace Matrix

Knowing Everything about the Requirements for a Computer-Based, Software-Intensive System

By Daniel M. Berry
University of Waterloo
dberry@csg.uwaterloo.ca
http://www.cs.technion.ac.il/~dberry

We are all aware of the problems with computer-based, software-intensive systems (CBSISs) not only these days, but also, since we first started to develop CBSISs. They do not do what they are supposed to do; they do not handle surprises well; they are not reliable; and they have poor user interfaces. We call them "brain damaged".

When we are designing a CBSIS, we have to think of everything, everything the CBSIS should do and everything the CBSIS should not do. Thinking of everything means figuring out what anyone using the CBSIS would want it to do under any circumstances and figuring out all possible stimuli, including multiple stimuli, to which the CBSIS must respond. In short, thinking of everything means figuring out all eventualities in all combinations. Like building architects and contract drafters, we CBSIS developers have to think of everything, even things of which no one in his or her right mind would have ever dreamt.

The real cause of any problem with CBSISs is really very simple. A problem happens simply because at no time in the development of the CBSIS to date, did anyone in the team that developed and maintains the CBSIS think of the problem long enough for action to be taken to solve that problem. By "team", I mean the entire set of people whose thoughts and actions affect the development and maintenance of the CBSIS. This group certainly includes all stakeholders, but can include also passers by who happen to interact with other members of the team in ways that affects the CBSIS's development. By "not thinking of the problem long enough", I am referring to the fact that many times, someone thinks of an issue briefly, but due to circumstances, including forgetfulness, sloppiness, and overloading, he or she fails to get the issue into the project's consciousness, often just by failing to write the issue down on a piece of paper. Consequently, the issue evaporates, never to be considered again until the CBSIS fails in a way that brings the issue to the forefront.

In other words, I am assuming that CBSIS developers really desire to avoid all problems and that once they are aware of a problem, they can in fact deal with it. Granted, there are really unsolvable problems, but by and large, with most CBSISs, being aware of a problem is a sufficient condition for its eventual solution.

Therefore, the solution to the problem of CBSIS failures is to make it possible for the team to think of everything. Clearly, we cannot expect any finite group of people in any finite amount of time to think of everything. In the words of Don Gause, no matter how much we succeed to learn about a CBSIS, there will always be that unknown issue that will rear its ugly head when the CBSIS is running; he calls these issues, "nature's last laugh".

Thus, we can expect that techniques that increase human participation in CBSIS development projects will help, while techniques that limit human participation will hinder our attempt to think of everything. We can expect that techniques that get people to think about CBSISs in a variety of new ways will help, while techniques that reduce thinking about CBSISs will hinder. Consequently, I am both pessimistic and optimistic about CBSIS development technology.

I am pessimistic when I see technology being touted as *the* solution to problems or as a way to avoid problems. I get particularly upset at method and tool evangelists who claim that their methods and tools will allow the users of the method to build better CBSISs because the method and tools themselves discover more problems. That is, they are promoting a reliance on the methods and tools and a way to reduce human involvement in the construction of the CBSISs. They claim that methods and tools will do the work for us. I just cannot see how any formal method or mechanized tool can help a designer discover a problem if the designer does not think of it. Indeed, concerns over following the method correctly may make it even harder to think of things not readily apparent.

I am optimistic when I see technology being offered as additional means to help the human developers see the CBSISs being developed under different lights, to apply human thinking and creativity, or to get different people involved in the projects applying the different technologies to the problems. In other words, as I have said elsewhere [1, 2], the biggest benefit of formal methods may be in bringing a group of people skilled in abstraction to work on the CBSIS, providing yet other views of the CBSIS, and attacking the problem with different kinds of thinking.

Therefore, I believe that the most important contribution that the field of requirements engineering is simply making people aware of the paramount importance of discovering and inventing requirements that deal with all issues that might come up. The next most important contributions of the field has to be a set of techniques, be they technical, managerial, social, or even psychological, to get more people into the development teams and to get them thinking more thoughts in more different ways about the CBSISs they are developing.

We have to help software engineers understand that just spending more time studying a problem and understanding its requirements before plunging into design and implementation pays off. We know that an error caught during requirements specification for a CBSIS costs two orders of magnitude less to fix than if it is caught after delivery [3]. We also know that devoting 25% as opposed to 0% of a total CBSIS development budget on the study phase reduces the cost overrun of the development from about 80% to about 5% [4].

Acknowledgments

The author thanks Don Cowan and Maria Nelson for comments on an earlier draft.

References

1. D.M. Berry, "Formal Methods, the Very Idea, Some Thoughts", in *1998 Monterey Workshop on Engineering Automation for Computer Based Systems*, Monterey, CA (October 1998).

2. D.M. Berry, "Formal Methods, the Very Idea, Some Thoughts on Why They Work When They Work", *Electronic Notes in Theoretical Computer Science* 25, Elsevier (1999). http://www.elsevier.nl/locate/entcs/volume25.html

3. B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ (1981).

4. K. Forsberg and H. Mooz, "System Engineering Overview", in *Software Requirements Engineering*, Second Edition, ed. R.H. Thayer and M. Dorfman, IEEE Computer Society, Washington (1997).

CORE-Blimey!

A regular column by Geoff Mullery, of Systemic Methods Ltd.

Enabling Network Systems

In previous contributions I have suggested that Harnden's Enabling Network System (ENS), implemented using features from Beer's Viable System Model (VSM) can assist the process of specification in a way current tools do not address. Here I illustrate this type of ENS and how it relates to the un-addressed problems of specification. No ENS of this type actually exists since this is simply my research concept, but technically all the capabilities for one do exist. Before I start, I must warn you that I mention here several past newsletter contributions (available from the RESG website http://www.cs.york.ac.uk/bcs/resg/).

The starting points for thinking about an ENS are the Internet and operating system user hierarchies. An ENS consists of a number of (potentially) linked Nodes, each containing key types of software facility which I shall describe in a moment. A Node is the point of presence of an individual or a group. A group node is like an operating system user id group in that it organises subsidiary nodes.

In an ENS a subsidiary node can be allocated to another node owner, so that the other owner can perform tasks in the role of that subsidiary node or in his/her separate, autonomous role. As the subsidiary node the things the user can do are constrained by the group owner, but as the autonomous node s/he is constrained only by legal and ethical restrictions.

With these capabilities it is possible to weave together a network (not just a tree) of interacting nodes, some of which have almost complete autonomy (like an individual user of the Internet operating under the domain name of an ISP) and others having partial or no autonomy (like the user of a company computer network with access to the Internet).

Any individual may have access to the ENS in one or more of several ways. First as an individual autonomous user; second as a constrained subsidiary user and third as an autonomous or subsidiary user of one group acting in the role of ("signing on" as) a subsidiary user of another group. Any node can set up a group (unless a parent group node denies the capability) and then becomes the owner of the group.

The owner of a group can grant read, write and execution permissions to other nodes in the same or other groups (again as long as s/he is not prevented by constraints imposed from a superior group node). When an ENS user is operating within a linked group node s/he is subject to the controls of that group.

What should be noted here is that there may be a network of service providers as there is now for the Internet, but hosted on that network is a logical network of nodes, some of which are group nodes – and group nodes are capable of applying group-specific controls.

Of course it is necessary to be much more precise about the characteristics of such a dual-networking concept in order to make it possible to build a practicable system, but space

does not permit me to go into more detail here. I merely appeal to the experience of readers with operating system and Internet experience to suggest that it is plausible for such an arrangement of nodes to be built.

The next question is, what are the "key types" of software which must be available to such nodes. Just letting people look at the data in other nodes is not communication – it is an invitation to chaos, as I believe the current Internet aptly demonstrates. There must be facilities for node subscription and cancellation (for the base service provider and for groups), group node control, publication of interests and needs, searching, filtering and translation between cultures or languages.

The primary thing a base service provider needs is to provide software for subscription to gain access to the hardware network, a base set of ENS tools which I shall describe in a moment, and facilities for storage of ENS inter-communication data structures. The latter is an extension of the idea of Web space – but not necessarily limited to the current Web site file formats.

The base set of ENS tools are then the equivalent of an extended Web browser, providing the node user with a facility to perform a number of tasks off-line and then go on-line to implement their application on the ENS.

The primary types of tool accessible within or from the browser are those which support the generalised communication and autonomy required by Harnden's ENS and some important sub-constructs evident in Beer's VSM. The VSM concepts of interest here are amplification (a distribution mechanism), attenuation (a collection mechanism), transduction (a transformation/translation mechanism) and co-ordination. In using these concepts I am interested here in what the VSM refers to as the control of variety – which for this purpose I interpret as the control of complexity for the ENS node user.

Support of amplification means supporting the node user in definition of needs and/or services s/he can provide and identification of controls on the propagation and duration of these definitions. Support for attenuation means support for definition of controls on the information returned by the ENS as a result of comparing the node user's needs and services with those defined by other node users.

Support for transduction means providing the node user with a means of investigating the possibility that the needs and services s/he has defined match those of some other node user, even though they have not expressed then in an exactly matching terminology.

Support for co-ordination means enabling the background discovery and updating of possible overlaps of interest between different nodes and then supporting the corresponding node-users in agreeing and establishing communication. Important distinctions here between the ENS and the current Internet are the manner and duration of the search and notification processes and the capabilities for communication.

The ENS must support the matching of different node users' needs/services over a period of time (defined by them) so that they do not have repeatedly to re-enter search criteria and reject things they have already seen. If something new turns up or there is a change which is relevant to an overlap of interest and the original definition is still "in-date", then the node user is notified of the new information.

The ENS must support much "cleverer" searches than is done by current Internet search engines. In particular it needs to take into account the possibility of differences in terminology between node users with a shared interest and it must support the ability to constrain where to search and from whence search requests and search answers are permitted to come.

The ENS must also support a much more flexible form of "bookmark" capability than the Internet currently allows. In particular I refer you to my previous contributions in issues 11 (Darn Those Threads) and 13 (What Can I Say?). The current trivial Internet browser bookmark facilities are totally inadequate for a complex information networking environment.

It must be possible for the reader to mark chains of references, not just single locations and it must be possible to categorise such chains with a purpose so that the reader need only see them if a matching purpose is selected. Also it should be possible for people to publish such chained sets of bookmarks so that other readers with a shared interest do not need to re-discover the chain.

The question of how all these facilities might be provided is not one I can answer in the space of an article like this. I will say however that from what I understand of the currently evolving software and hardware technologies there is nothing really standing in the way of provision of such capabilities – other than the fact that industry in general and the computer industry in particular appears to have little understanding of the fact that the solution to

RE-Bites...

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone." Kansas State Legislature, early 1890's

(Taken from: 'Collection of Ambiguous or Inconsistent / Incomplete Statements'
http://www.vuse.vanderbilt.edu/~jgray/funny.html)

main problems they really face reside in addressing precisely these issues.

The consequence is that we have a series of high powered analysis, translation, distribution and interconnectivity technologies all going to waste on solving problems which are really side-issues from the point of view of what causes problems in society in general and the society of system developers in particular.

With a basic set of facilities such as those I have suggested here, together with supporting the topics I discussed in newsletter issues 11 and 13 it is possible to provide for relief of the problem of people missing the existence of key information when it is actually available, while allowing the "owners" of key information to protect their property rights and grant limited access rights to other parties.

It is possible to let the existence of conflict persist and be made visible to those who need to know while still permitting a current "official" resolution of those conflicts to be made part of the formal agreed plan of work so that progress can be made – and those making progress can do it in the knowledge of where they must be careful because of a know conflict, with its inherent probability of future change.

Finally, it is possible to permit individuals to work efficiently in several roles, while constraining what they can do in selected roles to conform with a superior group requirement. The association of an individual with each role (node) makes it possible to detect possible divisions of interest and fore-warns of possible ulterior motives for policy decisions/recommendations by individuals.

These capabilities solve none of the problems mentioned but neither do they prevent the use of the currently available technical support tools for support of the technical problems most people in the computer industry ***think*** are the real problems.

What these capabilities do is make what I believe are the ***real*** problems (see newsletter issues 17 – All Together Now – and 19 – Political Requirements) easier to detect, track and treat and do these things more openly than is the case with current development environments.

There is one obvious additional question I have not addressed. That is the extent to which there must/can be a standardisation of the key ENS tools. In general my answer to that is that there should be as little standardisation as possible, but I have insufficient room in this contribution to justify that, so I will leave that debate to a later contribution.

RE-Publications

Book Reviews

Software Requirements - styles and techniques
Søren Lauesen
Samfundslitteratur, Copenhagen, 1999
ISBN 0-201-61593-2
Available from slforlag@sl.cbs.dk (approx £40)

Reviewed by Ian Alexander

Despite its title, this book discusses requirements in general, and most of it could perhaps have been called system requirements. It is quite distinctive in its approach, which combines industrial and academic experience. Most interesting are the real extracts from requirement specifications which form about 40% of the book.

The most original feature of the book is the idea of requirement "styles" mentioned in the subtitle. These are basically templates for individual requirements: for instance, the Feature style applies to "functional product properties described in plain text". An example is

> R1: The product shall be able to record that a room is occupied for maintenance in a specified period.

This is what other authors call a capability or function; similarly, the Performance style appears to be for a normal performance constraint. More distinctive, perhaps, are Defect and Subjective styles, which Lauesen suggests might be good for usability requirements:

> R2: On average, a novice user shall encounter less than 0.2 serious usability defects when …
> R4: 80% of users shall find the system easy to learn …

The issues around writing requirements in each style are discussed in terms of the risks to the customer and to the developer.

The book covers many basic topics effectively, and has a welcome amount of coverage on goals, tasks, issues, and use cases. Elicitation is rightly seen as something that not only involves many possible techniques – ranging from interviewing to QFD – but also that can be used to provide anything from a description of present work and problems to conflict resolution and completeness checks.

The book is not, and does not aim to be, a general textbook on Requirements Engineering. Instead, it seeks to introduce students – whether undergraduates studying for degrees, or system engineers on short training courses – to a range of practical techniques.

As Lauesen remarks, the book tries to "teach you many techniques, and advise you on when to use them and when not." The inherent danger in this approach is that it tends to present a subject as a maze of tangled paths and pitfalls. Dataflow, for instance, may not a good way of organizing requirements, but it is described nonetheless. Dataflow is rightly criticised in the book, since it either prejudges the software design (should a function become a module?) or makes tracing difficult when the functional decomposition fails to match the design decomposition. Guidance for beginning practitioners is thus weakened.

The book's origins influence it in many ways. Most noticeably, the presentation is a simple A4 format, with the 'slide' figures usually on the right hand page facing the text. Teachers are advised to spend "5 to 30 minutes on a single slide" with their students. This is quite a neat way to arrange study notes, but it results in large areas of white space. Perhaps for the same reason, there is no index or glossary. Helps are reduced to a single well-chosen page of bibliographic 'literature' which sketches the coverage of each of the referenced texts. There are exercises at the end of each chapter, but no answers are provided.

Lecturers and trainers faced with the same task as Professor Lauesen – to introduce groups of students to practical Requirements Engineering – may well find this a useful and likeable sourcebook. I would not expect many courses to use the book directly. However, there is much original thinking here, as well as an experienced teacher's careful classification of ideas and explanation of terms.

Real-life Examples

Practitioners and researchers may be most interested in the real-life examples which "have rarely been published." Lauesen discusses the reasons for this, including confidentiality, the difficulty of obtaining clearance, or that "the parties are somewhat ashamed of the specification style". As it is so unusual to see such examples, let us briefly review and compare them.

The first example is a 50-page specification for a business system for a Danish shipyard, carefully translated into English. It is complete except for the installation and contractual details. The requirements are unnumbered, relying bizarrely instead on page and line numbers. For example:

> 5.2.2.1.1 Debtor Management
> the debtor management system shall have the following functions:
> [Line 15] a) Data entry function for payments received and made in addition to invoicing made outside the contract system.

This approach cannot have made traceability easy.

The second example is "16 selected pages out of about 110" of the requirements for a public health administration system prepared by Deloitte and Touche – however did Lauesen get their consent? The requirements are untouched, which means they are in Danish, limiting

readership to those of us who have at least some knowledge of related germanic languages. The author claims that "The selected pages are a fair representation of the full spec. The full spec is just 'more of the same'." This claim is of course impossible to judge, without privileged access. The example contains the unforgettable requirement (I translate):

> The proposal shall be written in Danish.

The requirements in this example are mostly written in a remarkably familiar "Systemet skal …" idiom; they are numbered from 1 to 674 and have integer weights, e.g. (I translate):

> 56. It shall be possible to record from the telephone. (Weight: 2)

The third example is 4 selected pages out of about 20, for a noise source location system from Bruel and Kjær. This is in English supported by importance attributes, for instance:

> R-44 +++ It must be possible to repeat the measurement of a point at any time.

The requirements are set in context by three levels of headings and explanatory text before each item. The account is as fresh and direct as Lauesen's textbook.

Plainly there could be many fascinating comparisons of these intriguingly diverse examples.

Managing Software Requirements - a unified approach
Dean Leffingwell and Don Widrig
Addison-Wesley 2000
£37-99 (hardback)
ISBN 0-201-61593-2

Reviewed by Ian Alexander

This chunky handbook of almost 500 pages is a serious attempt at covering the whole field of Requirements Engineering from the viewpoint of Rational. Leffingwell is a VP of Rational and the developer of Requisite Pro, so traceability is unsurprisingly illustrated with matrices in that tool. Widrig is a consultant who trains Requisite Pro users, and the Series editors' names speak for themselves. That said, the book is not a sermon on the merits of Rational tools and methods, though there is an appendix on the Rational unified process which formalizes the book's approach. The book provides instead a wide-ranging and reflective account of the problems, people, processes and causes involved in putting requirements together. The style is direct and personal: ".. we described storyboarding ..", and the issues are discussed openly and practically.

The book is divided into 7 parts, rather Americanly named Team Skills. These are introduced with drawings, mercifully not clip-art, of people sitting at computer desks and thinking collectively into little clouds. The first one isn't a skill at all but the introduction which looks at the problem, the Requirements Engineering solution, and the need for software teams which Rational sees as including the requirements engineers. This is plainly one possibility, another being that a customer organization engineers the requirements and uses them to obtain what they want from a software house. The issue of whether there is anything to distinguish software from any other requirements is not raised.

The remaining Team Skills are analyzing the problem, understanding user needs, defining the system, managing scope, refining system definition, and building the right system. These titles are evidently somewhat artificial, but they broadly reflect the stages in a well-adapted approach to large system development: model the business, find out the user requirements, specify the system at high level, and make sure that what is built is what you asked for. The stages 'defining the system' and 'refining…' reflect the Rational idea of putting together an outline of a complex system in a 'vision document', essentially a high-level system architecture, then scoping this down to realistic proportions, and then writing software requirements and adding detail to the use cases.

The approach is definitely object-oriented but the authors are aware of other approaches and make an effort to be fair. For example, the chapter on 'technical methods for specifying requirements' in the 'refining the system definition' team skill section covers pseudocode, finite state machines, decision trees, flowcharts, ERDs, object analysis and DFDs. The authors rather grandly state that "each is worthy of an entire book on its own" – a bit of an exaggeration in the case of several of these techniques. Humour emerges, however, when comparing the effects:

"Data flow diagram models run the same risk as ERD models [being difficult for a nontechnical reader to understand, but ..] there is a larger danger in using DFDs in today's world; the OO folks will think that you are a 'functionally decomposing data modeler' and thereby fossilized matter and will ignore everything further you have to say on any subject."

At least it's clear which camp the authors are in; what is more, the descriptions are both short and readable, arguing the pros and cons of each technique. It is also good to see that even use cases are considered on their merits, and the authors recognize there are times when use cases aren't ideal. In particular, once use cases have been built up with additional details to serve as software specifications, they can become quite complex. Other techniques may be easier to understand.

The problem of illustrative examples is handled quite neatly with a fictitious case study for residential lighting automation. The case study is referenced throughout the book, and a nearly complete set of 'artifacts' - project documents - is provided in an appendix. The book's approach reveals itself as rather too heavy for the example project, suggesting a possible concern: no real account is

taken in the book of the needs of smaller projects. The appendix is however a useful and informative document in its own right, and would certainly be helpful for getting a project up to speed with object-oriented requirements management.

The book is well organized and indexed. The bibliography is, like the book, intentionally non-academic but contains a sensible list of practical and readable books on software, systems, requirements, objects, process modelling, and use cases. It is perhaps a slight pity that these topics are not itemized, nor are summaries provided.

Curiously, the authors do not define their target audience or supply a use case for the book. It is certainly aimed at the larger model of systems development, where subsystems are expected, and where software is dominant. Hardware is in fact mentioned, so the software in the title is not to be taken too seriously. The book contains no exercises for students, so the readership must be presumed to be practitioners who want to systematize their approach on larger projects. I think this book will prove quite helpful for that audience, as it describes a wide range of techniques accurately and clearly, in a practical step-by-step way, with a reasoned discussion of the pitfalls. They will not be dissuaded by the relatively high price, as the book contains much that is of value.

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: http://www.soi.city.ac.uk/homes/gespan/rq/rq.html

Web Pages

The web address of the RESG is:
http://www.cs.york.ac.uk/bcs/resg/

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

An interesting collection of 72 papers (!) and a description of an ESPRIT project on co-operative requirements engineering with scenarios. The CREWS project has developed two prototypical tool suites which can be employed, e.g., as extensions to the Use Case approach in object-oriented systems engineering. One shows traceable multimedia-based current-state analysis and animation of future scenarios, the other provides guidance for the creation and analysis of text scenarios and for the systematic generation of exception scenarios.

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

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

Mailing lists

The SRE list

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

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

RE-Actors

The committee of RESG

Chair: Dr. Bashar Nuseibeh, Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK. E-Mail: ban@doc.ic.ac.uk, Tel: 0171-594-8286, Fax: 0171-581-8024

Treasurer: Dr. Neil Maiden, Centre for HCI Design, City University, Northampton Square, London EC1V OHB, UK. E-Mail: N.A.M.Maiden@city.ac.uk, Tel: 0171-477-8412, Fax: 0171-477-8859

Secretary: Dr. Wolfgang Emmerich, University College London, Department of Computer Science, Gower Street, London WC1E 6BT, UK. E-Mail: W.Emmerich@cs.ucl.ac.uk, Tel: 0171 504 4413, Fax: +44 171 387 1397

Membership Secretary: David Shearer, Department of Computer Science, University of Hertfordshire, Hatfield, AL10 9AB, UK. d.w.shearer@herts.ac.uk.

Newsletter Editor: Dr Peter Sawyer, Lancaster University, Computing Department, Lancaster, LA1 4YR, UK. E-Mail: sawyer@comp.lancs.ac.uk, Tel: 01524 593780, Fax: +44 524 593608.

Associate Newsletter Editor: Ian Alexander, 17A Rothschild Road, Chiswick, London W4 5HS. E-Mail: iany@easynet.co.uk, Tel: 0181-995 3057

Publicity Officer: Carol Britton, Department of Computer Science, University of Hertfordshire, College Lane, Hatfield, UK. AL10 9AB, E-Mail: c.britton@herts.ac.uk, Tel: 01707 284354, Fax: 01707 284303

Associate Publicity Officer: Dr. Vito Veneziano, Department of Computer Science, University of Hertfordshire, College Lane, Hatfield, AL10 9AB, UK. E-Mail: v.veneziano@herts.ac.uk, Tel: 01707 286196.

Web-Master: Dr. Laurence Brooks, Department of Computer Science, University of York, York, YO10 5DD. E-Mail: Laurence.Brooks@cs.york.ac.uk, Tel: 01904 433242.

Industrial Liaison Officer: Dr. Barbara Farbey, Department of Computer Science, University College London, Gower Street, London WC1E 6BT, UK. E-Mail: B.Farbey@cs.ucl.ac.uk, Tel: o171 419 3672, Fax: 0171 387 1397.

Members-at-Large:

Dr. Steve Easterbrook, Department of Computer Science, University of Toronto, 6 King's College Rd, Toronto, Ontario, M5S 3H5, Canada. E-Mail: sme@cs.toronto.edu.

Dr. Sara Jones, Department of Computer Science, University of Hertfordshire, Hatfield, AL10 9AB, UK. E-Mail: S.Jones@herts.ac.uk, Tel: 01707 284370, Fax: 01707284303

RE-Creations

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

Deadline for next issue: 20th May 2000.