



Requireonautics Quarterly

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

© 1997, BCS RESG

Issue 12 (October 1997)

RE-Locations

<i>RE-Soundings</i>	1	A Historical Perspective on Requirements Engineering	13
Editorial	1	<i>CORE-Blimey!</i>	21
Chairman's Message	1	DB or not DB – That is <i>not</i> the Question.....	21
<i>RE-Day</i>	2	<i>RE-Publications</i>	23
Requirements Engineering: National Awareness Day.....	2	Book Review: "Requirements Engineering A good practice guide" by Ian Sommerville & Pete Sawyer.	23
<i>RE-Treats</i>	5	<i>RE-Calls</i>	24
Can the RE Process be Quality Certified?.....	5	Call for Viewpoints	24
Industrial Experiences in RE.....	5	Ninth IEEE International Workshop on Software Specification and Design (IWSSD9).....	25
The Unified Modelling Language (UML).....	5	5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems	26
Scenarios and use cases for requirements engineering:	5	<i>RE-Bites</i>	25
Business Process Re-engineering	5	<i>RE-Sources</i>	28
ACRE for Requirements Acquisition: A Tutorial	5	Web Pages	28
Distributed Requirements Engineering	5	Books.....	28
<i>RE-News</i>	5	Mailing lists.....	28
IEE Proceedings – Software Engineering	5	Tools.....	28
Calendar	6	<i>RE-Actors</i>	28
<i>RE-Readings</i>	7	<i>The committee of RESG</i>	28
Requirements Acquisition & Modelling: Impact of Reuse.....	7		
<i>RE-Papers</i>	9		
Quantifying The Qualitative: how to avoid vague requirements by clear specification language	9		

RE-Soundings

Editorial

Welcome to a special RE-day edition of Requireonautics Quarterly. You'll find a full programme for RE-day on the next few pages, after these messages. You'll also find a map and directions, so there'll be no excuses for missing it!

To celebrate RE-day, I'm delighted to be able to print two special articles in this issue. On page 9, you'll find an invited article by Tom Gilb, on Quantifying the Qualitative. If you've ever wondered how to make those fuzzy and evasive management directives more precise (and therefore attainable), then Tom's article will resonate. The second invited article, starting on page 13, is an historical overview of Requirements Engineering, by our very own Ian Alexander. Ian takes us on a journey through the last few decades, introducing the key players along the way, and best of all, allowing them to speak in their own words. Just perfect for that train ride on the 30th.

We also have our usual departments. There is another excellent column by Geoff Mullery (who will, of course, be espousing his viewpoint(s) from a soapbox at RE-day). We

have a review of the September RESG/REuseSG meeting (setting a standard for fast turn around that I know we'll have difficulty living up to next time...). And a review of Ian Sommerville and Pete Sawyer's latest book on Requirements Engineering.

Phew, we won't be able to get RQ through your letterboxes if it keeps growing at this rate. Still, I'm proud of the quality we have achieved in RQ over the last three years, and look forward to more. Keep those articles flowing in, and don't forget to send me contributions for RE-bites.

Copy deadline for the next issue is 19th December 1997, so that we can hit the streets early in the new year. So, until January, Merry Christmas!?!

Steve Easterbrook,
NASA IV&V Facility, Fairmont WV

Chairman's Message

Since this issue of the Requireonautics Quarterly is published on RE-Day, many of you readers may not be RESG members (yet). I therefore feel that I must use the first few sentences of my introduction to sell you the benefits of

membership...

In addition to this fine publication that you now hold in your hands, RESG membership entitles you to free attendance at the group's regular meetings, and discounted registration at many co-sponsored events. While not all events will be extravaganzas like RE-Day, the group aims to develop a programme of meetings based on your requirements. While we do our best to elicit these requirements whenever and wherever possible, we also need your feedback to validate our views and evolve them as your requirements change... Talk to us: requirements is our business!

OK, end of sales-pitch, now to RE-Day. Please remember though, I am writing this before the event and a lot of you - or is it a few - will be reading it afterwards.

We started thinking of organising an event like RE-Day over a year ago. The idea was to organise a get-together for current RESG members, and to attract and welcome new members into the group. To achieve this we needed a programme that would appeal to a wide range of people with diverse interests in requirements engineering and related disciplines. We think that our current RE-Day programme of 1 keynote, 3 tutorials, 7 workshops and a tools/publications

fair meets our original objectives. In case we missed anything out, we have provided a soapbox for attendees to stand on and be counted (courtesy of Philips Research Labs - thanks)!

Oh, and another thing. Attendance at RE-Day is free! To achieve this miracle, we delved into the RESG piggy-bank, which we have been prudently filling up since the RESG was launched three years ago. Madness I hear you say. Fear not: significant additional sponsorship has been secured courtesy of RENOIR - the European Network of Excellence in Requirements Engineering. We are grateful for its support.

Well, I hope you enjoy what's on offer in this newsletter, at RE-Day and at future meetings. Before I go however I must thank the RE-Day committee of volunteers for all their time and energy in organising the event. Thank you Andy, Carol, Cornelius, George, Ian, Mike, Neil, Olly, Sara, Shailey and Steve.

*Bashar Nuseibeh,
Imperial College, London*

RE-Day

Requirements Engineering: National Awareness Day

Tuesday, 30th September 1997, 9:30-5:30pm

Venue: King's Cross Holiday Inn (*see map next page*)

Cost: free to all

Everything you needed to know about requirements - and would like to ask! 'Requirements Engineering - National Awareness Day' (RE-Day) is the UK's premier Requirements Engineering event of the year. If you are involved in any aspect of Requirements Engineering, academically or industrially, then this is the event that you cannot afford to miss!

The purpose of RE-Day is to showcase the currently available best practice in Requirements Engineering. The day will contain a number of timetabled events interleaved with open tool vendor and other display sessions.

Programme

Requirements Tools Exhibition

09:00-17:00 Eleven leading vendors will demonstrate their requirements tools:

- Rational Software (Requisite Pro)
- ISDE (Tracer)
- QSS (DOORS)
- 3SL (CRADLE)
- Integral Solutions Ltd (PC PACK)

- SEA (TMA Toolset)
- Vitech Corporation (CORE)
- Bezant Object Technology (SEMATIC)
- ilogix (Statemate MAGNUM)
- Ascent Logic (RDD-100)
- TD Technology (SLATE)

Posters Exhibition

09:00-17:00 Posters from academic and non-academic organisations

09:00-17:00 'Requirements On The Go: An On-going Exercise In Collecting Requirements', Jill Hewitt, University of Hertfordshire

Keynote Address

09:00-10:30 'Requirements: Rough Sketch or Precise Specification' *Suzanne Robertson, Atlantic Systems Guild*

Soapbox

10:30-11:00 'Neither Fish, Fowl nor Good Herring: Requirements Engineering Deconstructed' John Dobson, University of Newcastle

12:30-13:00 'Just how important is requirements engineering?' Anthony Finkelstein, City University, London

15:05-16:00 'Alchemy is no substitute for engineering in Requirements Specification', Geoff Mullery, Independent Consultant

Tutorials

- 11:00 -12:30 'UML for Requirements Engineering' Jim Arlow, Logon Technology Transfer
- 13:30-15:00 'RAD and Requirements Engineering' John Crinnion, City University, London
- 15:30-17:00 'Requirements Engineering for Systems' Tom Docker, CITI Ltd

Workshops

- 11:00-11:45 'RE Training and Education' Linda Macaulay, Department of Computation, UMIST
- 11:45-12:30 'RE for Interactive Systems: Perspectives from the HCI Community' Gilbert Cockton (University of Sunderland) and Nigel Bevan (National Physical Laboratory)
- 12:30-13:30 'Unconventional Paradigms for RE alternative perspectives' Bernie Cohen, City University, London, Philip Boxer and Martin Loomes, University of Hertfordshire
- 13:30-14:15 'Overview of UK academic research projects' representatives from Oxford University, Lancaster University, University of York and University of Newcastle
- 14:15-15:00 'Overview of European academic research projects' representatives from non-UK European Research teams including Universite de Paris, Vrije Universiteit Amsterdam and Universite Catholique de Louvain
- 15:30-16:30 'Overview of UK Industrial Research and Practice' representatives from DERA, Nortel, GEC and Logica.

The RE-Day Keynote address:

"Requirements: Rough Sketch or Precise Specification?"

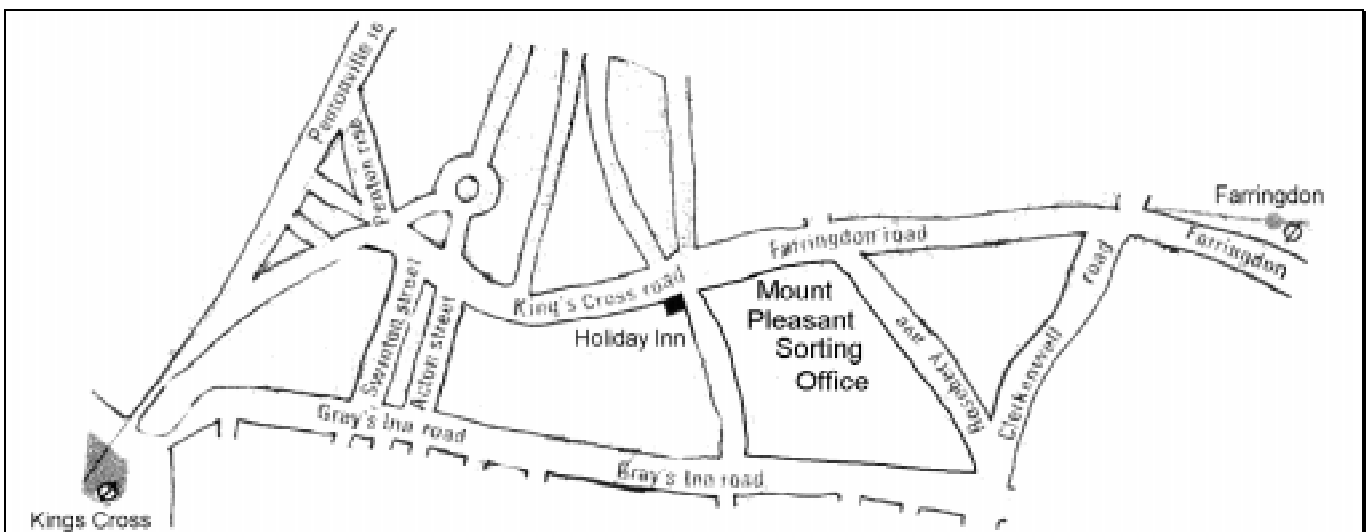
Speaker: Suzanne Robertson, Atlantic Systems Guild

Abstract: The role of a requirements engineer is to specify precise, consistent, testable requirements that provide a solid foundation for building the new system. However the source of the requirements, human beings, does not help to satisfy the goal of precision. Humans have different experiences, personalities, priorities and use of language, and these differences influence the way we state and hear requirements.

Given that precise statements of requirements are not the natural currency of human communication, it is folly for the requirements gatherer to assume that what he hears, especially the first version, is what is wanted. We can, and we must, tailor our requirements gathering to expect imprecision. Each fragment of requirement can be captured in the form of a rough sketch, and progressively nurtured and refined until it can be given all the attributes of a measurable and testable requirement. By fitting each requirement into a structure, it is far more convenient to progressively assess each requirement, and its interactions with other requirements, until it reaches its final stage of precision. By not immediately committing to a final version of the requirement, we are in a better position to react to the inevitable changes.

This talk discusses how to use the concepts of rough sketch, requirements template and requirements shell to mould imperfect wishes into a precise requirements specification.

About the speaker: Suzanne Robertson is a Principal of The Atlantic Systems Guild, an organisation renowned for helping people to improve the way they develop systems. Her



Directions from Farringdon Tube Station

Turn Right out of the station and walk up Farringdon Road. Cross Rosebery Avenue and go past the large Mount Pleasant Sorting Office which will be on your left. On the next corner is the Holiday Inn (Number 1, Kings Cross Road).

courses on systems analysis and requirements engineering have a world-wide audience. She is co-author of Complete Systems Analysis: the Workbook, the Textbook, the Answers (Dorset House, 1994), a two-volume text and case study that teaches the core skills necessary for systems analysis. Suzanne is successfully researching and consulting on patterns and the specification and reuse of requirements. The product of this research is Volere, a complete requirements process and template for specifying requirements and assessing requirements quality.

Professional affiliations include membership of IEEE and the Australian Computer Society, the committees for Object Technology, International Conference on Requirements Engineering 98 and International Conference on Software Reuse and Roving Ambassador for the British Computer Society's Reuse Group

RE-Day Mini-Tutorial 1: UML for Requirements Engineering

Tutor: Jim Arlow

The Unified Modelling Language (UML) of Booch, Rumbaugh and Jacobson is defining the strategic direction for object-oriented methods. This tutorial focuses on those aspects of UML which may be effectively used in Requirements Engineering. In particular, it presents Use-Cases, Scenarios, Class Diagrams, Sequence Diagrams and Collaboration Diagrams which are the artefacts generated in the Analysis phase of the OO project lifecycle. Basic concepts and UML syntax will be discussed, and the mapping of high level system requirements (Use Cases) down to individual Classes will be demonstrated.

About the tutor: Dr. Jim Arlow has been involved in OO methods since 1990 and is currently constructing the Enterprise Object Model for British Airways. When not working for BA, he is a trainer and consultant for QA Training, Rational and LogOn Technology Transfer. Highlights of his career include winning an award at Object World Frankfurt for the best OO development environment and helping to introduce object technology to BA. Jim has also done training and consultancy for Mannesmann, Standard Life, DEC, IBM and Burlington Northern Santa Fe Railroad.

RE-Day Mini-Tutorial 2: RAD and Requirements Engineering

John Crinnion, City University

This mini-tutorial will examine the principles and practice of rapid and evolutionary systems development methodologies, and will discuss how they address the requirements engineering aspects of business applications development. The tutorial will comprise three parts:

1. Principles and definitions: What is RAD, RAD & structured systems development, RAD & evolutionary development, RAD & O-O, The DSDM Methodology

2. Video, illustrating RAD in practice
3. The RE implications, both theoretical and practical: Converging ideas in systems development.

About the tutor: John Crinnion is a senior lecturer at the City University in London. He is an experienced practitioner, who has been writing and speaking on the subject of RAD and evolutionary development since the late 80's. He has worked extensively as a consultant on RAD projects of all sizes, and is the author of the book 'Evolutionary Systems Development' (published by Pitman 1991).

RE-Day Mini-Tutorial 3: Requirements engineering for systems

Tutor: Tom Docker, Director, CITI Limited

Requirements engineering for systems is much more than capturing the problem. It is essentially managing the expectations of large numbers of stakeholders to deliver products with clear business benefits, when a system can be in development for a significant number of years (e.g., a new railway service).

A characteristic of many systems is that component subsystems have different rates of development and maturity, which introduces a potentially critical timing dimension to producing an adequate overall solution. A relatively long development life frequently means that the profile of claimed benefits change and that, if the problem is not adequately described, the requirements themselves are liable to change.

A further area of concern is the likelihood that different subsystems will, in reality, be provided by different 'subcontractor' groups. Consequently, traceability and configuration management become key factors.

In this tutorial, key issues are discussed related to the identification and structuring of requirements and their mapping to adequate solutions, to ensure that monitoring of the products and benefits can be maintained throughout the product development lifecycle.

About the tutor: Dr Thomas Docker has worked in the IT function of a number of commercial and government organisations. He taught computing and IT at Massey University in New Zealand. Tom was also managing director of a consultancy company that helped determine management strategies and business processes. He is the author of numerous papers related to project management, the specification of products, and management education. He is director of the postgraduate programme in business process management, and is a visiting fellow of the University of Bradford. He has helped a number of large government and commercial organisations to review and modify the way that they specify and manage products, and has particular interest in the specification of requirements related to benefits realisation.

RE-Treats

Forthcoming events organised by the group.

Can the RE Process be Quality Certified?

Wednesday November 26, 1997

Location: Room 418, Huxley Building, Imperial College, London

Time: 2:00pm to 5:00pm

Cost: free to members, £5 to others

This meeting will debate the question of whether it is currently either feasible or desirable to attempt quality certification of the RE process. Can such an exercise be meaningful in a climate in which the community's view on RE 'best practice' is evolving on a yearly basis? The debate will be led by:

Ian Sommerville, Lancaster University
Richard Stevens, QSS Ltd

Industrial Experiences in RE

Wednesday February 4, 1998

Location: York University

Time: 2:00pm to 5:00pm

Cost: free to members, £5 to others

Format: Invited speakers.

The Unified Modelling Language (UML)

March 25th 1998

Details to be announced

Scenarios and use cases for requirements engineering: How helpful are they?

May 1998

Location: City University, London

This one-day meeting will investigate how best to use scenarios and use cases for acquiring and validating system requirements. There has been considerable recent interest in scenarios and use cases as a starting point for object-oriented

analysis and design. However, the role and usefulness of scenarios and use cases for requirements acquisition and validation is less clear. The meeting will explore how best to use scenarios and use cases during requirements engineering processes. Presenters will include practitioners, method and software tool vendors, and academic researchers. Presentations will cover current experiences, available software tools and methods, and recent research results.

For more information contact Dr Neil Maiden; tel: +44-171-477-8412; e-mail: N.A.M.Maiden@city.ac.uk.

Business Process Re-engineering

June 98

Details to be announced

ACRE for Requirements Acquisition: A Tutorial

September 1998

Speakers: Dr Neil Maiden and Dr Gordon Rugg

This half-day tutorial will inform attendees of a range of techniques drawn from a range of disciplines that are available for requirements acquisition. It draws on ACRE, a framework for selecting and using different requirements acquisition techniques. The tutorial will present the framework and give practical guidelines for using some effective but less-known techniques.

Distributed Requirements Engineering

November 98

Details to be announced

Please note: RESG's policy is that distribution of speakers' slides will be free of charge to those who attend the meeting and specifically request a copy. Those people who do not attend the meeting and request a copy of the slides will be asked to pay £5 to cover photocopying and mailing costs.

RE-News

IEE Proceedings – Software Engineering

The BCS/IEE Software Engineering Journal has now been superseded by IEE Proceedings - Software Engineering. Editors are John McDermid, Jeff Magee and Ian Sommerville. Problems during the transition period have meant that many people did not submit papers, and so there is no backlog of papers waiting to be published.

The editors would like to encourage everyone to submit to the journal and make it a leading international software engineering journal of the same status as IEEE Transactions. We will do all we can to speed up the reviewing process so

that papers submitted now have a good chance of publication early next year. Papers, of course, must be original but we welcome extended versions of high-quality conference papers that have been presented.

Papers should be sent to
Mary Howley,
IEE Proceedings - Software Engineering
Michael Faraday House,
Six Hills Way
Stevenage SG1 2AY

Calendar

October 1997

OOPSLA '97 Workshop, Atlanta, Georgia, USA, October 6, 1997. <http://www.forsoft.de/~rumpe/oopsla-ws/>

10th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW-97), Sant Feliu de Guixols, Catalonia (Spain), October 15 - 18, 1997. <http://www.acia.org/ekaw>

The 1st International Enterprise Distributed Object Computing Workshop (EDOC'97), Marriott Resort, Gold Coast, Australia, October 24-26, 1997. <http://www.dstc.edu.au/events/edoc97/>

The Second Australian Workshop on Requirements Engineering (AWRE'97), Macquarie University, Sydney, Australia, 27th Oct 1997. <http://www-comp.mpce.mq.edu.au/~didar/awre97.html>

November 1997

The 8th International Symposium on Software Reliability Engineering (ISSRE'97), Albuquerque, New Mexico USA, November 2 - 5, 1997. <http://admin.one2one.com/issre97>

IASTED International Conference on Software Engineering, San Francisco, USA, November 2-5, 1997. <http://www.iasted.com/sanfran/se97.html>

International Workshop on (Situating) Method Engineering, Delhi, India, 3-4 November 1997. *Info: cvs@sms.utwente.nl*

12th IEEE International Conference on Automated Software Engineering (ASE'97) (Formerly the Knowledge-Based Software Engineering Conference [KBSE]), Hyatt Regency Lake Tahoe, Nevada; USA, November 3 - 5, 1997. <http://ic-www.arc.nasa.gov/ic/conferences/ASE97>

Fourth International Symposium on Software Metrics (Metrics '97), Albuquerque, New Mexico USA, November 5 - 7, 1997. <http://www.cs.pdx.edu/conferences/metrics97/>

Fourth International Conference on Information and Knowledge Management (CIKM'97), Baltimore, Maryland, USA, November 8-11, 1997. <http://enws396.eas.asu.edu/cikm97/cikm97.html>

4th International Conference on Object-Oriented Information Systems (OOIS'97), Brisbane, Australia, 10-12 Nov 1997. <http://www.cs.uq.edu.au/conferences/oois97>

Symposium On Current Trends In Software Engineering, Louvain-la-Neuve, France, November 15, 1996. <http://www.info.ucl.ac.be/event/CTSE.html>

The Second International Workshop on CSCW in Design, Bangkok, Thailand, Nov 26-28, 1997. <http://www.chinavigator.co.cn/edu-sci/cscwd97.htm>

December 1997

Joint 1997 Asia Pacific Software Engineering Conference (APSEC) and International Computer Science Conference (ICSC) Hong Kong, 2 - 5 December 1997. <http://www.comp.hkbu.edu.hk/~apsec>

January 1998

Engineering Complex Computer Systems Minitrack, part of the Emerging Technologies Track of the 31st Annual Hawaii International Conference on Systems Sciences (HICSS), Big Island of Hawaii, HI, January 6-9, 1998. <http://www.ce.unipr.it/hicss>

8th Workshop on Knowledge Engineering: Methods & Languages (KEML'98), Karlsruhe, Germany, January 21-23, 1998. <http://www.aifb.uni-karlsruhe.de/WBS/dfe/keml98/index.html>

February 1998

Second International Workshop on Development and Evolution of Software Architectures for Product Families, Las Palmas de Gran Canaria, Spain, February 26 & 27, 1998. <http://hvp17.infosys.tuwien.ac.at/ARES/>

March 1998

Second Workshop on Formal Methods in Software Practice (FMSP98), Clearwater Beach, Florida, USA, 4-5 March 1998. <http://www.bell-labs.com/user/maa/fmosp98>

European Joint Conferences on Theory and Practice of Software, Lisbon, Portugal, March 30 - April 3, 1998. <http://www.di.fc.ul.pt/~llf/etaps98/>

International Conference and Workshop on Engineering Of Computer Based Systems, Jerusalem, Israel, March 30 - April 3, 1998. <http://www.ece.arizona.edu/~ecbs/ECBS-98.html>

Empirical Assessment & Evaluation in Software Engineering (EASE'98), Keele University, Staffordshire, UK, 30th March - 1st April 1998. <http://www.keele.ac.uk/depts/cs/Announcements/conferences/ease98.html>

April 1998

Third IEEE International Conference on Requirements Engineering (ICRE'98), Colorado Springs, Colorado, USA, April 5-10, 1998. <http://www.cs.technion.ac.il/~icre98/Ninth>
IEEE International Workshop on Software Specification and Design (IWSSD9), Ise-shima, Japan, April 16-18, 1998. <http://salab-www.cs.titech.ac.jp/iwssd9.html>

The Eleventh Workshop on Knowledge Acquisition, Modeling, and Management (KAW'98), Banff, Alberta, Canada, April 18-23, 1998. <http://ksi.cpsc.ucalgary.ca/KAW/KAW.html>

The 20th International Conference on Software Engineering (ICSE'98), Kyoto, Japan, April 19-25, 1998.
<http://icse98.aist-nara.ac.jp/>

The 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC '98), in conjunction with The 4th IEEE Workshop on Object-oriented Real-time Dependable Systems (WORDS '98) and The 4th International Workshop on Object-Oriented Real-Time Systems (WOORTS '98), Kyoto, Japan, April 20 - 22, 1998. <http://dream.eng.uci.edu/isorc/>

June 1998

"Modelling and Design". 5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'98). Abingdon, UK, June 5-8, 1998.

<http://www.dcs.qmw.ac.uk/research/hci/dsvis98>

International Conference On Formal Ontology In Information Systems (FOIS'98), In conjunction with the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 6-8, 1998. <http://mnemosyne.itc.it:1024/fois98/>

IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '98), Stanford University, CA, June 17-19, 1998.
<http://www.cerc.wvu.edu/WETICE/>

Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE '98), June 18-20, 1998 at the Hotel Sofitel, San Francisco Bay, USA.
<http://www.ksi.edu/seke/seke98.html>

September 1998

5th International School and Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98) Lyngby, Denmark, Sept 14-15 (School) and Sept 16-18 (Symposium) 1998. <http://www.it.dtu.dk/~ftrtft98>

September 1999

FM'99: Formal Methods 1999, The World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, Late September 1999.
<http://www.it.dtu.dk/~db/fm99/>

RE-Readings

Reviews of recent Requirements Engineering events.

Requirements Acquisition & Modelling: The Impact of Reuse

The last meeting of the RESG was an afternoon Panel Session, organised jointly with the BCS Software Reuse Specialist Group, held on Wednesday, 10th September 1997 at UMIST.

Report by **Ian Alexander**

This was a very interesting meeting, consisting of four short but quite different talks, with serious discussions after each and a stimulating panel discussion afterwards. The committee would like to thank everyone who took the trouble to come up from London to be present.

The meeting was chaired by Neil Maiden (City University, and RESG Treasurer), who asked a series of thought-provoking questions:

- whether reuse was feasible or desirable?
- whether we wanted to reuse components, patterns, or knowledge?
- whether reuse was to take place during requirements capture, or elsewhere?
- whether reuse was to be within or across domain (or product) families?
- and whether there were unseen advantages of reuse, such as negotiation?

The speakers were David Corral (with Andrew Daw) of GEC Marconi, Mike Mannion of Napier University, Wing Lam of the University of Hertfordshire, and Mark Jones of InFact Systems.

David Corral described an approach towards reuse of large-scale patterns in systems engineering, including not just requirements but design and verification as well. At the scale of warship development, for example, the challenge is to manage the complexity of many interacting systems and subsystems. Tens of thousands of requirements have to be followed through the entire life-cycle. System engineering consists of modelling so as to achieve a cohesive set of requirements. It might be possible to reuse entire patterns (including design and verification) with a total system engineering approach. Clearly a common purpose is essential: the large consortia involved must not be adversarial in nature.

The process is always iterative, involving many feedback loops based on trade-off studies, and it was such studies that could, with care, be reused effectively. The focus of modelling moves from static (theoretical) models including logical (the required operations), physical (functions, architecture) and implementation (equipment), down to dynamic (empirical) models, which could consist of "black box" descriptions of external performance or of "white box" details of internal performance. These inevitably return actual test results to the static models, which are in turn modified, and which then pass improved hypotheses back to the dynamic models for further testing. The interactions are non-linear, so the big question is how to do trade-offs well.

Corrall believed that the quality of reuse would depend on the consistency of the chosen viewpoints on to systems and subsystems. These would examine goals, and constraints on them, referencing common resources and a shared decision engine. In this context, reuse could only be of patterns perceived within a project. A reuse library to store patterns would be a key capability; for success, it would be essential to measure the cohesiveness of such patterns. Measurement depends in turn on a total system engineering environment, while risk is controlled by a proper understanding of interactions between components. It would not be sensible to try to reuse requirements alone.

Mark Jones described progress in a difficult environment, namely the high-pressure world of financial trading. His project, at Barclays de Zoute Wedd (BZW), was to implement a completely new office system for 4000 PCs, covering a choice of 1200 applications organized into no fewer than 100 different configurations for 3800 people. To make matters worse, the move was to take place in just 8 months, meaning that a major office move would happen every fortnight! The office being replaced made use of 70 different kinds of PC, and the range of configurations had become unmanageable with spiralling ownership costs. The approach was to

- standardize the PCs to a single model to enable reuse;
- to lease the PCs to cut the hardware ownership costs;
- and therefore to have a PC desktop which could easily be rebuilt when an individual moved to a new machine.

The chosen platform was Windows-NT, offering a very limited choice of tooling (MS-SMS) which “did not work on this sort of scale”. The requirements “evolved all the time” also, so the audience had no trouble appreciating why Jones said that the traditional waterfall lifecycle model was not adequate. The usual approach would have been to repeat the software builds, overlapping them to save time, but basically starting afresh on each new requirements document. This would have solved the same problems many times over, a very inefficient approach. Instead, Jones arranged for a small number of skilled technicians to create an HTML catalogue of components, which could readily be browsed, and a matching library of the actual software. The team soon discovered that reuse demanded “much more detail in the catalogue” than was usual for software.

Success in the venture was proven by the results: at the end, Jones could build desktops two or three times as rapidly as at the start. All the catalogue and library tools were “home-brewed”, based on an Access database and a custom browser. The main difficulty was to enforce and police the rules: there was an understandable tendency for overloaded support staff to make a quick-and-dirty fix for a defective PC, without documenting it, leading to further problems later. Users (low in the BZW hierarchy) found they did not get what they asked for, going rather against the grain of PC use where everyone has their own favourites on their “own” desktop, and also going against the technology (“dear me, the technology”) which encouraged customization. But by

the end, 90% of all components were being effectively reused, and most of the users were compliant. Reuse on later projects would allow Jones to bid at a lower price than his rivals, locking the client into the contractor. Reuse therefore always had “political” or financial overtones.

Mike Mannion presented MRAM, the method for requirements engineering and management, a study for the operations centre of the European Space Agency (ESOC). The idea was to discover ways of reusing User and perhaps also System Requirements from specific systems within whole application families. All mission planning systems might be alike, but they were also all different: “different projects overlap in complex ways” and there were typically unique “discriminants” separating any chosen pair of projects. A “Viewpoint” is “an encapsulation of partial knowledge of an application domain from a single (stakeholder’s) perspective”, enabling reuse of a block of requirements. Effectively, the study proposed a meta-model tool, with a database of all ESOC systems. The tool was based on Access, Rational Rose, and Word seamlessly integrated, to manage natural-language requirements. The database could be browsed on the World Wide Web, enabling the vital 30 or 40 people scattered across Europe (and places as far away as Canada) to study a specification. The obvious problem is that reuse (with a domain model) has little attraction for the contractors working at ESOC: it might threaten their profits, and they might justifiably plead increased project risk if requested to reuse someone else’s components. Reuse may only cut costs on a relatively extended time-scale: NorTel suggest it might be on the 5th reuse, for example. Plainly there is a large difference between a commercial manufacturer of satellite systems (eager to reuse everything from one model to the next) and a research organisation where each system is by definition new. But in these cost-conscious days, even organisations funded by governments must attempt to use money wisely.

Lam combatively presented some excellent arguments against the possibility of requirement reuse, and was if anything disappointed to find that the panel of speakers largely agreed with him: not that reuse was impossible, but that it certainly did not make sense at the level of atomic requirements. He argued that reuse

- is a non-purist view of RE (which starts from user requirements);
- introduces bias, such as assumptions made on earlier projects;
- depends on “fragile commonalities” (Jeff Kramer, 1993);
- is impossible when requirements change;
- runs counter to the ethnographic approach, which demands analysing problems with an open mind to users (e.g. Hughes, Randall & Shapiro, CSCW ‘92);
- skimps on proper analysis!
- depends on an as-yet non-existent RE process model;
- interferes with the customer/supplier relationship;

- has little real impact on costs, unless it manages to reduce testing as in the case of safety-critical systems.

Lam drew a graph showing the inverse relationship between the number of copies of a system and the negotiability of the requirements. An ESOC-style one-off development has “freely negotiable requirements” and reuse is difficult; Rolls-Royce style product variants have “semi-negotiable”

requirements, with some measure of reuse, while real software products have non-negotiable requirements and reuse is not only possible but normal. Maybe, Lam wickedly suggested, when reuse is successful it always gets renamed as “infrastructure” or “generic software”.

RE-Papers

Quantifying The Qualitative: how to avoid vague requirements by clear specification language

By Tom Gilb, Senior Partner, Result Planning Ltd.

Overview

Qualitative requirements can and should always be specified as clearly as possible. Most people do not know how. Everybody can learn how. Organizations should insist on quantitative clarity for all critical qualitative aspects of their project, process or product. The main concept is to define an ‘operational scale of measure’ for all qualitative requirements.

Current Culture

We have all seen requirements of the type:

Change: {enhance, improve, reduce, increase}

Something: {performance, ease of use, maintenance costs, productivity, security, adaptability}.

For example:

“substantial increase in our product’s ease of use by the customer”.

Such statements are sometimes even called “non-functional requirements”, which is evasive. They are ‘quality requirements’. They are *not* functional requirements (*what* the system will do), they are not *cost* requirements (which *resources* the system will use), nor are they *design ideas* (what we shall *do* to make the system meet its requirements). They are also not *general constraints* (which create a fence around the permitted area of requirements and design).

For certain limited purposes, such as emotional management speeches or hiding the real objectives, these vague requirements statements can be useful. For serious projects, where large investments, long duration, high impact results and high risk are frequently involved, this primitive culture is unnecessary and unacceptable. For example, when one project’s quality requirements were quantitatively documented, it was discovered that an order of magnitude difference over the required level of cost reduction existed between what the senior managers expected and what the project manager was aiming to deliver.

The quality levels required should drive the search for solutions. That is, you need to understand the magnitude and

nature of the change required to select the potential solutions and even, to assess whether the change is realistic within the given resource constraints.

One top manager I worked with (Robb Wilmott, then Managing Director of ICL) immediately realised the advantages of quantifying quality requirements. He turned around the same day (in about 1983) to his direct reports and said they would get no more money or power from him without quantification of all the wonderful things that they had been promising him. That led to a culture change at the top and, I would like to think contributed, to a company that went into profit and stayed there, while almost all competitors ran up losses. But, the first thing I had to do was to tutor the directors on how to quantify their glorious futures. Not one of them had any previous training or culture in doing so.

I would recommend that specifying quality requirements quantitatively is taken up to the most senior levels. However, it can be applied at any level, it is effective and applicable to your current project or process. Its use is certainly not restricted to just software engineering. In one company (Douglas Aircraft, in 1987-9), I was enlisted to teach the engineering managers and senior engineers “How to Quantify the Unquantifiable”. They were motivated when they saw it was possible. But their engineering training, management training and company culture had never shown them how to quantify qualitative ideas in *general*.

The situation is probably similar at your company. The motivation is there, but the ‘know how’ is not.

There are several companies that already have clear top management policies about quantifying quality requirements and objectives (e.g. ICL, Ericsson, Hewlett-Packard and IBM). Maybe your company has as well - you probably need to dig into the company standards to see if they exist. But most companies have not taken a position on the subject. And even when they have, the top management wisdom is not always backed up by a pervasive company culture, and necessary training.

Well if you want an interesting mission in life, this is your opportunity. The rest of this paper will assume that you would *like* to know how to quantify qualitative ideas and will concentrate on the *how*.

Expressing a Variable Idea as a defined Scale of Measure

Let me show you a set of ideas for describing a qualitative idea.

“reduce weight”
can be expressed as

<p>Weight: Gist: to lose enough weight so that I don’t feel overweight. Scale: Weight in relation to average weight for height, age and sex expressed as percentage. Meter: weighing scales compared to tables of normal weight. Past [Last Year, Me] 130% Must [End this Calendar Year, Me] 110% Plan [End Next Calendar Year, Me] 100% <- New Year Resolution</p>

This is a ‘requirement statement’ using ‘Planguage’ (Planning Language), a defined requirements definition language that I have developed over the past twenty years.

It is fairly obvious exactly what the requirement is by just reading it. And since we were dealing with a well-known quantitative idea, weight, you should have had no intellectual problems with the quantification. Did you notice that this specification tells you more than ‘reduce weight’? There are actually many distinctly different interpretations that you can get from such a vague requirement. To maximise your probability of achieving the goal and avoid wasting resources, you need to home in on the ‘correct’ intended one. Maybe you will need several attempts before you get the correct version agreed with your customers. In fact, the weight specification could be improved if I added information on my weight, height and age. For example, age could have an impact on designing the solutions.

Well, if we can specify our requirements in this format, people will understand a lot better what we mean. All you have to do is copy the pattern above. Table 1 gives some explanation of the formally defined language used above. Did you notice that each statement could have several components?

Parameter Name	Qualifier	Level on Scale	Source of the level
Plan	[End Next Calendar Year, Me]	100%	<- New Year Resolution

The parameter name is a predefined concept in the Planguage. It has a precise meaning and function.

The qualifier(s) distinguish between different required levels at different times, places and under different conditions [when, where, if]. They allow you to specify requirements in three basic dimensions of time, space and event. A requirement does not have to be a simple point, it can be a curve altering over time. This allows requirements to be stated for both the short term and the long term. And there can be several dimensions of these curves. This allows you to differentiate requirements for different customer types, users and products. You can also state requirements conditional upon certain events or certain conditions, being true.

For example:

Plan [End 1998, My weight, If working overseas during 1998 > 90% of the time] <105% <- TG’s Best Guess
--

The level on the Scale is where we numerically express what we know about the requirement. This does not mean we

Example’s Parameter	What’s this?	Used for	Note also
Weight	Name of requirement	Cross referencing, reuse of concepts	Reduces need to repeat full requirement in many places
Gist	A rough informal idea of the requirement	Summarising, getting consensus	Useful departure point, but rarely a clear definition
Scale	Definition of the requirement’s scale of measure	Getting precision and clarity. Defining the concept	Contractual use
Meter	Definition of how we are going to measure or test the attribute in practice	Agreeing as to how the requirement fulfillment will be judged in practice	Contract
Past	A benchmark of past status for that requirement	Knowing the meaning of concepts like “improved”	Useful reference point
Must	A future requirement target which is necessary for system survival	Early delivery minimum levels. Trade-off minimum levels	Contract minimum payment level
Plan	A future requirement target which is necessary for success and satisfaction	Understanding the full requirement. Knowing when to stop designing and building	Contract full payment level

Table 1: Explanation of the elements of Planguage

have to know the precise correct value. ‘Greater than 20%’ or ‘20% +/- 5%’ or ‘over 20 years old’ can be sufficient, so do not get hung up on providing a value. Make a start: specify a level, consider its accuracy and the source of the information and, if you are unsure, decide whom you should ask to get a better value. Maybe, the value can only be obtained by testing in the field and will need to be refined over time.

Note, the level specification has no meaning without us also knowing the defined ‘scale of measure’ and the ‘qualifiers’.

Finding Scales of Measure for less obvious requirements

Now, if we can just apply these ideas to all qualitative requirements we would have pretty clear requirements ideas. But a critical problem remains. If I say weight: you think {Pound, Kilos, Tons, Grams, Ounces}. For some concepts we have immediate cultural access to one or more acknowledged scales of measure. Often with internationally standard definitions.

But, for most of the requirements we will deal with in real life, we cannot think of a scale of measure at all. Nothing readily comes to mind. And we are not trained to ‘develop’ the scales of measure. We do not have a catalogue to look them up in. We do not have a culture that demands that we find them. So, we give up. We make excuses: ‘it is qualitative’, ‘it is not measurable’ ; all of which means that we are unnecessarily defeated in getting control over the concepts. They have won the battle and vagueness can ravage our systems at its leisure.

These are concepts like

- ‘easy to maintain’
- ‘portable’
- ‘secure’
- ‘user-friendly’
- ‘safe’
- ‘adaptable’

Yet surely there must be some approach to quantifying these concepts? We are interested in them because if they are at ‘bad’ levels, they cost us time and money. So, it is ‘in the cards’ that there must be *some* way to express how much damage or good the different degrees of them can cause us.

And, a little common sense will get you to acknowledge that we could, for example, use scales of measure like the simplifications below:

<p>Maintainable: Scale: how fast we can fix something.</p> <p>Portable: Scale: how much effort to move to a new environment compared to total new construction there.</p> <p>Secure: Scale: the % of defined classes of detected attacks which would penetrate the system in defined ways.</p> <p>User-Friendly: Scale: The time needed for defined people to master defined tasks.</p> <p>Safe: Scale: the probability that defined dangers will result in defined bad results.</p> <p>Adaptable: Scale: the time or effort needed to adapt defined systems to defined new requirements.</p>

Derivations of Basic Scale Definitions

Many of these are obvious when stated, but we may never have seen them before, so we have some difficulty ‘inventing’ them. However there seem to be a few basic patterns of scales of measure, and everything else seems to be a dialect or derivation. I have defined many of the basics in more detail in my book ‘Principles of Software Engineering Management’ (PoSEM) and the reader is encouraged to copy and make these ideas available to themselves and their colleagues.

Continuous Improvement on your Catalogue of Basic Scale Definitions

Once you have derived your scales of measure for your organization’s projects, you will find that the work you have already done can usefully be stored and retrieved for future projects. Sometimes it takes a lot of work to ‘craft’ a really good scale of measure. And some experience to know if it is good and useful. So we need to be in a systematic continuous learning mode about these ideas.

A Defined Process for Finding Scales of Measure for Quality Requirements

In my free Web publication “Requirements-Driven Management” (RDM, Section 4.4) there is a slightly more detailed rigorous process defined for quantifying quality, and the entire book gives all manner of supporting detail. But I shall give you the essence here and now:

1. Use your common sense to try to work out a scale of measure.
2. Look up ideas for similar scales of measure in books and handbooks, including your own company collection of such scales of measure from previous projects.
3. Tailor the definition to your current purpose, using [generic scale definitions] and corresponding [qualifiers] in the target requirements statement.
4. If the concept is complex, and no suitable scales seem to cover what you want control over, then consider ‘exploding’ the concept into sub-concepts until you reach a level of detail that you can identify scales. Then proceed to step one above for each of the elements of your definition’s explosion.

Generic Scale Definitions

One method for making scales of measure definitions more-easily reusable is to define them ‘generically’ and let the necessary detail come out elsewhere, for example, in the target {Plan and Must} statements. Separate definitions can also be used. There are two main classes of separate definitions:

1. Defined terms (A: Defined: a definition of A.)
2. Qualifiers (which themselves may be formally defined), e.g. in Plan [1999, USA] 60%, ‘USA’ has a predefined meaning.

For example:

Maintainability

Gist: ability to fix faults

Scale: the clock time for [1. defined Class of faults] from
 a [2. defined Start point, Default: first knowledge of the fault by any system or person]
 to
 [3. a defined Completion point OR Default: the point where it is in fact correctly fixed, without any negative side effects and this is verified]
 by
 [4. defined fixing and testing Process OR Default: any process]
 using
 [5. defined Capability of person OR Default: any one who does the job in practice]
 during
 [6. defined System Development Phase]
 RF: Requirements Faults: Defined: any requirement specification judged by the Inspection process as having a Major defect, i.e. a defect likely to lead to major errors downstream (in the design, test and/or field phases) with high costs to correct.
 P1: { Class = RF,
 Start = Detection Logging in Inspections,
 Completion = Approved in Inspection,
 Process = Fixing in Master Specification,
 Capability = Requirements Author}
 P2:[defined System Development Phase]
 FT: First Release into Field Trial.
 GPRI: First General Product Release Internationally:
 Defined: after field trials, the date of paid access by more than one country to our product.
 Past [P1, P2 = FT] Average 24 hours <- Inspection Statistics
 Plan [P1, FT] 10% of Past
 Must [P1, GPRI] 1% of Past <- Project Manager

Hopefully, if you study the above example, it will give you some ideas. Notice:

1. The scale is extremely generic, but it does have automatic defaults, which are reasonable.
2. There are six generic parameters which can be used to define the Scale's meaning
3. These six parameters can be specified in target (Must, Plan) and benchmark (Past) statements. They can be reused any number of times in the scope of a single overall requirement definition (Maintainability).
4. A useful set of these parameters can be defined once, as 'P1' above and reused, to simplify and clarify (that the same set is intended).
5. 'Names' or 'labels' can be applied not only to the overall requirement (like 'Maintainability') but also to any useful sub-component of the definition (as in the example 'P1', 'P2').
6. Any parameter concept (e.g. 'First General Product Release Internationally') can be formally defined locally (as in this example) or globally, say in a Project

or Requirements Glossary. The general signal that a term has a formal definition somewhere is given by 'Capitalization' of the words describing it.

What I am showing you in this short article should be sufficient to inform you of the possibilities for better definition of not only quality requirements, but any requirements. The above examples only hint at the richness of the 'Planguage'. I encourage you to get your free copy of my RDM book from the web sites. You will even find ready-made defined standards for specification, which you can incorporate into your company standards and teachings.

Principles for Quality Quantification

I will leave you with a set of 10 of the 100 stated principles from the RDM book (RDM, Section 4.5) :

0. THE PRINCIPLE OF 'BAD NUMBERS BEAT GOOD WORDS'

Poor quantification is more useful than none; at least it can be improved systematically.

1. THE PRINCIPLE OF 'QUALITY QUANTIFICATION'

All qualities can be expressed quantitatively, '*qualitative*' does not mean unmeasurable.

2. THE PRINCIPLE OF 'MANY SPLENDORED THINGS'

Most quality ideas are usefully broken into several measures of goodness.

3. THE PRINCIPLE OF 'SCALAR DEFINITION'

A scale of measure is a powerful practical definition of a quality.

4. THE PRINCIPLE OF 'THREATS ARE MEASURABLE'

If lack of quality can destroy your project then you can measure it *sometime*; the only discussion will be 'how early?'.

5. THE PRINCIPLE OF 'LIMITS TO DETAIL'

There is a *practical* limit to the number of facets of quality you can define and control, which is far less than the number of facets that you can *imagine* might be relevant.

6. THE PRINCIPLE OF 'METERS MATTER'

Practical measuring instruments improve the definition of 'scales of measure'.

7. THE PRINCIPLE OF 'HORSES FOR COURSES'

Different quality scale meters for different times and places will be necessary.

8. THE PRINCIPLE OF 'BENCHMARKS'

Past history and future trends *help* define words like "improve" and "reduce".

9. THE PRINCIPLE OF ‘NUMERIC FUTURE’

Numeric future *requirement* levels complete the quality definition.

References

PoSEM: Gilb, T., “Principles of Software Engineering Management” (1988, 13th printing 1997), Addison-Wesley Longman. This book has many chapters on quantification of requirements and related subjects.

RDM: Requirements-Driven Management, a book manuscript with 100 slides found at <http://www.stsc.hill.af.mil/SWTesting/gilb.html>

Homepage: <http://www.Result-Planning.com/>
This Gilb home page will give direct access to or URL pointers to detailed Gilb literature (papers, slides, book manuscripts). It includes RDM and my newest book on “Evolutionary Project Management”, which also incorporates the requirements definition ideas given in this paper.

Author Biography

Tom Gilb was born in the USA on Dec. 24 1940. He emigrated with his family to London in 1956, and then went to Norway in 1958, where he joined IBM and, studied Sociology and Philosophy. He maintains a second residence in Covent Garden, because it is near the Ballet. He is the author of 8 published books and has two manuscripts awaiting publication. More detail via the Homepage <http://www.Result-Planning.com>



A Historical Perspective on Requirements Engineering

By Ian F Alexander

Summary

This article takes a look at the background of requirements engineering, using examples from the computer science literature. Extensive quotations from named original sources are included to illustrate how thinking has developed.

Introduction

The chosen examples, while diverse in style and content as well as in epoch, together give an impression of a very particular kind of progress: a movement back through the systems development life-cycle towards the user. The dates of the stages along the braided stream of this movement are to a large extent arbitrary. I have included at least one extract to support each date, though both earlier and later sources could be found.

Early systems, especially software (where most of the complexity now lies) were focused on the machine as a scarce and expensive resource.

Attention gradually moved, in software terms, from code to design, and then on to specification. This was understood initially as the precise description of components-to-be-built; gradually this understanding too broadened to encompass entire systems.

Finally, with input from the human-centred sciences (psychology, sociology, ethnology...) specification has come to include a definition of the problem to be solved, as seen by the human users of any putative system. Even the term “users” contains within it a trace of the old assumption that the system, the virtual machine, is the centre of attention: in Julian Hilton’s ringing phrase, the user is a computer peripheral.

With the current emphasis on dialogue and human-computer (or human-machine) interaction, the movement can be seen to be continuing towards a proper balance between people and systems.

Several major trends in technology have driven this progress in understanding the place of requirements in development:

- falling price and increased accessibility of computer-based systems
- growing interactivity, bringing a widening range of users with rising expectations
- increasing memory and program size, bringing problems of complexity
- rising size and cost of system failures despite ever-better development tools.

Any one of these trends could have had a powerful impact on development methodology. Together they have forced system and requirements engineering to transform themselves into full engineering disciplines.

The Architecture of Complexity (1962)

It is an odd paradox that the principles underlying the management of complex systems were clearly stated *before* the software crisis, the chaos that results when complexity runs riot during development. Herbert A. Simon wrote his paper *The Architecture of Complexity: Hierarchic Systems* (Proceedings of the American Philosophical Society, 106, Dec 1962, 467-482) while computers were still scarce, and programs for them short by today’s standards. Simon has spoken out clearly on the logical structure of systems for much of the 20th century:

“One very important class of systems has been omitted from my examples thus far: systems of human symbolic production. A book is a hierarchy in the sense in which I am using that term. It is generally divided into chapters, the chapters into sections, the sections into paragraphs, the paragraphs into sentences, the sentences into clauses and phrases, the clauses and phrases into words. We may take the words as our elementary units, or further subdivide them, as the linguist often does, into smaller units. If the book is narrative in character, it may divide into “episodes” instead of sections, but divisions there will be.

“The hierarchic structure of music, based on such units as movements, parts, themes, phrases, is well known. The hierarchic structure of products of the pictorial arts is more difficult to characterize, but I shall have something to say about it later”.

Plainly, Simon could have added Software to the list of classes of complex systems produced symbolically by humans, as the next section of his paper indicates.

“Let me introduce the topic of evolution with a parable. There once were two watchmakers, named Hora and Tempus, who manufactured very fine watches. Both of them were highly regarded, and the phones in their workshops rang frequently - new customers were constantly calling them. However, Hora prospered, while Tempus became poorer and poorer and finally lost his shop. What was the reason?”

“The watches the men made consisted of about 1,000 parts each. Tempus had so constructed his that if he had one partly assembled and had to put it down - to answer the phone, say - it immediately fell to pieces and had to be reassembled from the elements. The better the customers liked his watches, the more they phoned him and the more difficult it became for him to find enough uninterrupted time to finish a watch.

“The watches that Hora made were no less complex than those of Tempus. But he had designed them so that he could put together subassemblies of about ten elements each. Ten of these subassemblies, again, could be put together into a larger subassembly; and a system of ten of the latter subassemblies constituted the whole watch. Hence, when Hora had to put down a partly assembled watch to answer the phone, he lost only a small part of his work, and he assembled his watches in only a fraction of the man-hours it took Tempus’.

“It is rather easy to make a quantitative analysis of the relative difficulty of the tasks of Tempus and Hora: suppose the probability that an interruption will occur, while a part is being added to an incomplete assembly, is p . Then the probability that Tempus can complete a watch he has started without interruption is $(1 - p)^{1000}$ - a very small number unless p is 0.001 or less. Each interruption will cost on the average the time to assemble 111 parts (the expected number assembled before interruption). On the other hand, Hora has to complete 111 subassemblies of ten parts each. The probability that he will not be interrupted while completing any one of these is $(1 - p)^{10}$, and each interruption will cost only about the time required to assemble five parts.

“Now if p is about 0.01 - that is, there is one chance in a hundred that either watchmaker will be interrupted while adding any one part to an assembly - then a straightforward calculation shows that it will

take Tempus on the average about four thousand times as long to assemble a watch as Hora.”

Hierarchical structuring controls the complexity of programs, of designs, and of specifications. It is sad that the software industry had to go through so much pain to rediscover what was already known.

The Software Crisis (1972)

The high price of computers created a virtual priesthood of technically-minded specialists who knew how to build and maintain large systems. Enid Mumford's book *Systems Design: ethical tools for ethical change* (Macmillan, 1996) is a study of and a reaction against the technocentric thinking that characterised the first generation (perhaps 1955 to 1985, for the sake of argument) of software developers. She quotes Harold Sackman (Mass Information Utilities and Social Excellence, Auerbach, 1971) as explaining “the ideology of technical systems analysts of the time:”

“Early computers were virtually one of a kind, very expensive to build and operate. Computer time was far more expensive than human time. Under these constraints, it was essential that computer efficiency came first, with people last.... Technical matters turned computer professionals on; human matters turned them off. *Users were troublesome petitioners somewhere at the end of the line who had to be satisfied with what they got.*”

With this unhappy phrase, Sackman surely captured the unspoken thought of many technicians. Here is part of Mumford's account of the development process in that period:

“Because systems design has been defined as a technical activity, traditional methods have been tools and procedures for designing technical systems. The systems analyst has not seen his role as helping with the management of complex change. He has restricted this to providing technical solutions.

“Because of this narrow view the practice of systems design has generally been broken down into a number of sequential operations. Typically, these have included analysis - gaining an understanding of the problem that has to be addressed and describing the activities, data and information flow associated with it. This led to a requirements definition. Next came a specification or description of the functions to be performed by the system to process the required data.

“Design followed, and this covered the development of the internal structure of the software which would provide the functions that had been specified. Implementation was the development of the computer code that would enable the system to produce data. Validation checked that each stage was successfully accomplished and ‘evolution’ was the correction of errors or modification of the system to meet new needs.

“Until recently, the human user of the system figured very little in this technical approach. Consideration was not given to issues which are regarded as of prime importance today - business goals, needs and structures, competing demands for information, multiple interest groups and dynamic and complex business environments.”

Clearly, if users were such a nuisance, it is hardly surprising that their needs were not often met. In case you are minded to believe that negative attitudes to users are a thing of the remote past, reflect on this short extract from Steve Maguire’s *Debugging the Development Process* (Microsoft Press, 1994):

“When Microsoft first began conducting usability studies in the late 1980s to figure out how to make their products easier to use, their researchers found that 6 to 8 out of 10 users couldn’t understand the user interface and get to most of the features. When they heard about those findings, the first question some programmers asked was “Where did we find eight dumb users?” They didn’t consider the possibility that it might be the user interface that was dumb. If the programmers on your team consciously or unconsciously believe that the users are unintelligent, you had better correct that attitude - and fast.”

Perhaps the classical text of the crisis era is *Structured Programming*, (Academic Press, 1972) edited by Tony Hoare. The title is worth reflecting on: the best minds of the day were focused on the problem of *making* programs, i.e. on the actual implementation; the idea that the real problem lay well before that phase took another decade to arrive. The book’s first section, the modestly-entitled *Notes on Structured Programming* is written by E. W. Dijkstra, which he introduces as follows:

“ON OUR INABILITY TO DO MUCH

“I am faced with a basic problem of presentation. What I am really concerned about is the composition of large programs, the text of which may be, say, of the same size as the whole text of this chapter. Also I have to include examples to illustrate the various techniques. For practical reasons, the demonstration programs must be small, many times smaller than the “life-size programs” I have in mind. My basic problem is that precisely this difference in scale is one of the major sources of our difficulties in programming!

“It would be very nice if I could illustrate the various techniques with small demonstration programs and could conclude with “..and when faced with a program a thousand times as large, you compose it in the same way.” This common educational device, however, would be self-defeating as one of my central themes will be that any two things that differ in some respect by a factor of already a hundred or more, are utterly incomparable. History has shown

that this truth is very hard to believe. Apparently we are too much trained to disregard differences in scale, to treat them as “gradual differences that are not essential”. We tell ourselves that what we can do once, we can also do twice and by induction we fool ourselves into believing that we can do it as many times as needed, but this is just not true! A factor of a thousand is already far beyond our powers of imagination!

“Let me give you [an] example to rub this in. A one-year old child will crawl on all fours with a speed of, say, one mile per hour. But a speed of a thousand miles per hour is that of a supersonic jet. Considered as objects with moving ability the child and the jet are incomparable, for whatever one can do the other cannot and vice versa.

“To complicate matters still further, problems of size do not only cause me problems of presentation, but they lie at the heart of the subject: widespread underestimation of the specific difficulties of size seems one of the major underlying causes of the current software failure. To all this I can see only one answer, viz. to treat problems of size as explicitly as possible...

“To start with, we have the “size” of the computation, i.e. the amount of information and the number of operations involved in it. It is essential that this size is large, for if it were really small, it would be easier not to use the computer at all and to do it by hand. The automatic computer owes its right to exist, its usefulness, precisely to its ability to perform large computations where we humans cannot. We want the computer to do what we could never do ourselves and the power of present-day machinery is such that even small computations are by their very size already far beyond the powers of our unaided imagination.”

Complexity was for Dijkstra the major issue. There is no indication that he had heard of Herbert Simon’s work; nor that he thought questions of *problem* definition important.

We cannot end this section without at least a small extract from perhaps the most famous description of the software crisis, Frederick P. Brooks’ *Mythical Man-Month* (Addison-Wesley, 1975). The “tar-pit” is a masterpiece of rueful expression of experience, and it illustrates the book’s cover.

“No scene from prehistory is quite so vivid as that of the mortal struggles of great beasts in the tar pits. In the mind’s eye one sees dinosaurs, mammoths, and saber-toothed tigers struggling against the grip of the tar. The fiercer the struggle, the more entangling the tar, and no beast is so strong or so skillful but that he ultimately sinks.

“Large-scale programming has over the past decade been such a tar pit, and many great and powerful beasts have thrashed violently in it. Most have

emerged with running systems - few have met goals, schedules, and budgets. No one thing seems to cause the difficulty - any particular paw can be pulled away. But the accumulation of simultaneous and interacting factors brings slower and slower motion..."

It took at least another twenty years before the most important factor was seen to be accuracy of specification of requirements. But Brooks was also far ahead of his time in his understanding of the importance of consulting users when writing the "specifications" for a system:

"The manual, or written specification, is a necessary tool, though not a sufficient one. The manual is the *external* specification of a product. It describes and prescribes every detail of what the user sees. As such, it is the chief product of the architect.

"Round and round goes its preparation cycle, as feedback from users and implementers shows where the design is awkward to use or build... The manual must.. describe everything the user does see, including all interfaces.. but must not attempt to dictate the implementation."

This is fascinating both for its clarity and for its confusion. The principles of iterative development, of system life-cycle, of interface definition, of modularity, of review, and of independence of specification from design, are all clearly enunciated in these few lines. At the same time, there is no distinction drawn between user manual and system specification (though Apple Corporation have long made a virtue of writing the user manual in advance, and using it as the specification); nor, more seriously, is there any separation between user and system requirements.

The Structured Response (1980)

The first practical response to the so-called Software Crisis was the introduction of structured methods, first to design and later to analysis (of requirements). Meilir Page-Jones' *Practical Guide to Structured Systems Design* (Yourdon Press, 1980) was one of the key books of the movement. Until then, designers had largely ignored the problem of defining software requirements:

"The designer does not talk directly to the user; the analyst is the go-between linking the designer and the user. On one side, he has to converse with the user to elicit exactly what system the user needs. On the other side, he has to communicate the user's requirements to the designer so that the designer can plan a computer implementation of the user's system. To do this, the traditional analyst writes a document called a functional specification.

"Many systems analysts were once designers or programmers, and are more familiar with the world of EDP and with its jargon than with the user's world and his business jargon. The result of this has all too often been doubly disastrous: First, the user is bewildered by a specification buzzing with EDP

terms - such as disk drives, job steps, and HIDAM - whose meaning he can only guess at. Yet the user is expected to peruse, correct, and finally accept this specification as being a true representation of his needs.

"Second, the designer, whose job it is to decide the best implementation for the user's business system, is pre-empted by the analyst in his decisions. The analyst may frustrate the designer by imposing premature and often arbitrary constraints upon him. Indeed, in many a shop where I have worked or consulted, the user community and the design/programming community are two peoples separated by a common document - the functional specification.

"Structured Analysis is a response to the fact that a specification that improperly records the user's needs, more than any other single factor, is likely to jeopardize a project's success."

In this clear and lively style, Page-Jones persuaded a generation of software designers to produce decent dataflow diagrams and data dictionaries before building their systems. The problems didn't all go away, though.

The Waterfall Model (1984)

The structured approach clearly implied a development life-cycle that ran analysis ... design ... build ... test ... operate. The most basic interpretation of this cycle is that each activity or phase happens exactly once, starting when the previous one has finished. The process was visualised as a waterfall splashing down from one rock (analysis) to the next (design) until the product appeared at the bottom.

One of the clearest statements of the model remains the European Space Agency's software engineering standards, which were first issued in 1984 as BSSC(84)1, but are now better known as PSS-05 (Issue 2, 1991). Behind the dry and repetitive style of a standards document, the reader can perceive the real concern to improve the structure and quality of often mission-critical software:

"UR phase: user requirements definition

"The UR phase is the 'problem definition phase' of a software project. The scope of the system must be defined. The user requirements must be captured. This may be done by interview or survey, or by building prototypes. Specific user requirements must be identified and documented in the User Requirements Document (URD).

"The involvement of the developers in this phase varies according to the familiarity of the users with software. Some users can produce a high quality URD, while others may need help from the developers.

"The URD must always be produced. The review of the URD is done by the users, the software and

hardware engineers and the managers concerned. The approved URD is the input to the SR phase.

“SR phase: software requirements definition

“The SR phase is the ‘analysis’ phase of a software project. A vital part of the analysis activity is the construction of a ‘model’ describing ‘what’ the software has to do, and not ‘how’ to do it. Building prototypes to clarify the software requirements may be necessary.

“The principal deliverable of this phase is the Software Requirements Document (SRD). The SRD must always be produced for every software project. Implementation terminology should be omitted from the SRD. The SRD must be reviewed formally by the users, by the computer hardware and software engineers, and by the managers concerned...”

The Formal Methods Argument

Vagueness, inaccuracy and ambiguity have long been perceived as major problems in specifications. Bertrand Meyer’s *On Formalism in Specifications* (IEEE Software, Jan 1985, 6-26) attempted to solve the real problems of writing requirements by moving to the precise language of mathematics:

“A critique of a natural-language specification, followed by presentation of a mathematical alternative, demonstrates the weakness of natural language and the strength of formalism in requirements specifications

Meyer identified ‘seven sins of the specifier’, which he listed as follows:

“Noise: the presence in the text of an element that does not carry information relevant to any feature of the problem.

Silence: the existence of a feature of the problem that is not covered by any element of the text.

Overspecification: the presence in the text of an element that corresponds not to a feature of the problem but to features of a possible solution.

Contradiction: the presence in the text of two or more elements that define a feature of the system in an incompatible way

Ambiguity: the presence in the text of an element that makes it possible to interpret a feature of the problem in at least two different ways

Forward reference: the presence in the text of an element that uses features of the problem not defined until later in the text

Wishful thinking: the presence in the text of an element that defines a feature of the problem in such a way that a candidate solution cannot reasonably be validated”

Every requirements engineer will recognise these ‘sins’ as ever-present dangers (though forward reference is not necessarily a sin if used explicitly). They are avoided by careful wording, discussion with users, and most importantly review. The text can be supported where necessary by

diagrams and (in technical domains) by equations. They cannot be replaced by specifications written entirely in set-theoretic or logical notation, for the sufficient reason that users (and probably also programmers) cannot be expected to understand it.

A deeper reason is that given by Einstein in his aphorism

“As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.” (quoted by Fritjof Capra, *The Tao of Physics*, Wildwood 1975).

For real-time systems, Einstein’s aphorism means that no mathematical model can be perfect, for it would have to model the entire universe (in which all particles are connected by forces and quantum effects). For example, turbulent flows over the wing of an aircraft can be described well but never perfectly by simulation and modelling. Hence a formal specification of such a system cannot hope for mathematical exactness. This does not mean that precision is not worth striving for, in the interests of safer and better systems.

Meyer is unusual in recognising quite clearly the difference between the problem space and the solution space (as in his definition of overspecification, above). Unfortunately, his proposed formalisms apply in large part to precise specifications of design features, such as “sequences”, which belong usually to the solution space. For example, the maximum line length of a sequence *s* is defined as

$$\text{max_line_length}(s) = \max (\{ |j-i| \mid 0 \leq i \leq \text{length}(s) \text{ and } \text{forall } k \text{ in } i+1 \dots j, s(k) \text{ new_line} \})$$

This kind of low-level detail is far closer to code than to users, even if it is argued that it constitutes a specification rather than a solution. In fact, even the assumption that there will be a fixed line-length is part of a solution: as a counter-instance, lines on Web pages are as long as users make them when they resize browser windows and change font sizes. I would therefore assert that all such detail is a priori likely to be part of a design rather than part of a problem statement. The underlying user requirement for Meyer’s sequences might be “Text must not disappear off the right of the screen”, or “Text should wrap normally”, if indeed it was expressed at this level at all. Formal methods have a definite and necessary place in system engineering, but that place is in the (later) solution phases of the life-cycle.

Even practitioners in real-time safety-critical systems design have moved towards this point of view. Martyn Thomas, for example, in an article *Can Software Be Trusted?* (Physics World, Oct 1989, 30-33), wrote:

“...if you can monitor more variables and take action quickly on what you find, you can control your process so that it is nearer its optimum efficiency. With cruder control mechanisms you may have to allow much greater safety margins, or to tolerate many more occasions when the controlled process is shut down by the safety system, either of which would cost a lot of money.”

Thomas has here identified the driving force in real-time systems design: improved automation saves money. For example, aircraft can be made lighter and more fuel-efficient, but at the same time more dependent on their control systems. He goes on:

“Unfortunately, programmable systems also bring novel problems of design, verification and assessment...

“[The] specification may be wrong. It may inadequately reflect the task which, with hindsight, we would have wished the computer system to perform. If the specification is contradictory, we should be able to detect it with analysis techniques; if it is incomplete, we may have enough information to discover where (for example, it may not specify the required behaviour for all possible input values). If, however, the specification is functionally wrong, we can only hope to realise it by prototyping or testing (or by some other form of review by experienced humans). Otherwise our implementation will seek to satisfy the given specification as accurately as possible, with the result that the computer system will be ‘correct’ (with regard to the specification) but the overall system behaviour will be wrong.

“When society asks how safe a system is, what can we say?”

Thomas’ perspective is important as it sets limits on the effectiveness of design methods. Even if these were perfect, safety-critical systems such as aircraft would continue to fail. Thomas is a definite proponent of formal methods, in their proper place:

“It is mathematics which models precisely the nature of digital systems, and it offers a scientific basis for the development of computer systems. These methods are the way of the future, offering far greater certainty throughout the development process than the traditional ‘craft’ methods can provide.

“However, formal methods do not address the whole problem, because there is clearly still the problem of understanding the controlled process, developing an adequate specification, analysing the system for potential hazards and the consequences of failure, and so on. And, of course, even a mathematical analysis or proof has some probability of being incorrect. Formal methods should be viewed as a way of strengthening the development process.”

The Object-Oriented Challenge

In a famous paper, *Structured Analysis and Object-Oriented Development are not Compatible* (ACM Ada Letters, Nov/Dec 1991, 56-66), Donald Firesmith argued powerfully against the mixing of techniques, and claimed that functional analysis was obsolete:

“It is the thesis of this paper that traditional Structured Analysis, even when modified for real-time systems, is a technically obsolete way to specify

software requirements, and that Structured Analysis and Object-Oriented Development are fundamentally incompatible and unnecessarily difficult for software engineers to combine effectively.”

I believe that this claim is based on a failure to distinguish between user and system (or software) requirements. It is certainly true that it is awkward to move from a set of (functional) dataflow diagrams describing a system to a set of objects implementing that system. Firesmith believes, with reason, that it is better to describe the system as a set of objects, identifying interfaces between such objects rather than between functions.

The approach breaks down, though, with user requirements. Users cannot immediately identify objects that will persist throughout the system life-cycle. They must first describe the problem which the system has to solve, before anyone can sensibly propose a system, even in outline, which might be decomposed into objects which can solve the problem. For example, it has been argued for the case of a traffic automation system that objects such as “Traffic Signals” and “Crossroads” can be assumed to persist throughout development. This confuses the solution (a crossroads, safeguarded by automatic signals) with the problem (to get cars and people across a road safely). Objects such as signals cannot be identified in the problem space, because they do not exist there. Alternative solutions such as roundabouts (circular junctions with no lights) or highway junctions (overpass/underpass with slip roads) address the same problem with entirely different objects.

It is thus not easy to see how object-oriented analysis could ever be extended to cover the start of the system life-cycle, the user requirements phase. As for the question of whether object-oriented methods will completely supersede functional analysis in system and software requirements phases, time will tell. Now, several years after Firesmith’s paper, the methods coexist. Object-orientation has delivered excellently on some projects and poorly on others; projects that plan to use objects in design do well to use them for system analysis also.

The Ethnological Approach

As with all trends, it is hard to put a precise date to the use of ethnological techniques in system development. The classical text is perhaps H. Garfinkel’s *Studies in Ethnomethodology* (Prentice-Hall, 1967), though Enid Mumford’s sociotechnical work, including her ETHICS development method (1979 and later) may have been more influential.

A fascinating paper on the use of ethnology in the development of air traffic control systems, *Faltering from Ethnography to Design* (John A. Hughes, David Randall and Dan Shapiro, CSCW 92 Proceedings, 115-129, Nov 1992, © ACM 1992; extract used by permission of the Association for Computing Machinery) shows how the approach has deepened since the 1980s, and the problems that remain:

“The aim of this paper is to explore some ways of linking ethnographic studies of work in context with the design of CSCW systems. It uses examples from an interdisciplinary collaborative project on air traffic control. Ethnographic methods are introduced, and applied to identifying the social organization of this cooperative work, and the use of instruments within it. On this basis some metaphors for the electronic representation of current manual practices are presented, and their possibilities and limitations are discussed.

“There is an almost universal agreement in CSCW on the desirability of interdisciplinary design. While the integration of cognitive psychology in HCI substantial body of achievements on record, the objective now is to incorporate perspectives on the social organization of work - at a theoretical but, more importantly, at a practical level - into the canon. With some exceptions, however, these remain pious hopes, or else independent analyses of work at some remove from the context of real system design. This paper makes a modest contribution to realising the goal of integration. It does so by taking a particular design project: a prototype system for air traffic control; and by carrying an ethnographic analysis of the social organization of the work involved through to specific implications for the design of a computer-based system to support that work...

“The [air traffic controller’s cardboard] strip turns out to be a vital instrument for ATC. It is possible, and on occasion necessary, to perform ATC with the strips but without radar. To do so with radar but no strips, however, would pose extreme difficulties. When a controller gives an instruction to a pilot, for example to ascend to flight level 220, s/he marks this on the strip: in this case, with an upwards arrow and the number 220...

“Our work has now reached a stage where we are generating system interfaces whose design has been informed by the ethnographic observations. We have found that the information provided by ethnography is essentially background information which has provided a deeper understanding of the application domain. The ethnography did not result in specific, detailed systems requirements; rather it provided pointers to appropriate design decisions...

“As well as influencing the systems design process, we believe that the ethnographer has a further role as substitute user during initial system validation. User-centred design where users participate in the interface design process from an early stage in that process is likely to lead to more effectively usable user interfaces. However, a serious constraint in the practice of user-centred design is the availability of users, particularly when these users are an expensive

and scarce resource. This is a particular problem during early stages of the design process where the design is unlikely to satisfy the user’s requirements so the user gets little reward from participation...

“Software engineers and sociologists can work together effectively. However, there is a wide gulf between these disciplines and entrenched philosophical positions will probably ensure that that gulf cannot be bridged. Effective inter-disciplinary cooperation requires much flexibility on both sides and requires both sides to question their own assumptions and working methods.”

The Scenario Approach

The idea of using scenarios, physical sequences of activities actually carried out to achieve some desired goal, is surprisingly recent. The idea plainly has its origins in the Taylorian time-and-motion studies to improve efficiency in early 20th century mass-production factories. It is possible that a reaction against Taylorism has delayed the use of scenarios in discovering and prioritising requirements.

A text devoted entirely to the approach is *Scenario-Based Design*, edited by John M. Carroll (Wiley, 1995). Despite its title, it provides quite a rich discussion of the use of scenarios in requirements analysis and dialogue between developers and users: both critical topics. The chapter *Rapid Prototyping of User Interfaces Driven by Task Models* by Peter Johnson, Hilary Johnson and Stephanie Wilson introduces the subject in this way:

“In the fictional country of *Erewhon*, Samuel Butler presents many scenarios of life and work in a society in which people serve the needs of machines rather than machines serving the needs of people. In his novel Butler warned of the dangers of isolating the design and development of technology from people and society, and from any considerations of the effects of technology on people’s lives. While in *Erewhon* the technology was that of steam engines and mechanical devices, the serious consequences of focusing design solely on machines rather than on people’s needs, and how those needs might be best served by the design and development of machines, were clearly envisioned. People became the servants of machines. The task of looking after the machines became the prime focus of work, with little consideration for subsequent effects on the quality of the work experience or indeed the quality of the end product. The design and development of technology to improve the quality of work and the quality of the products of work require us to pay close attention to the nature of the work, and to be explicit about how any technology that we design might affect people and their work. The mistake is to believe that computer system designers are only responsible for the software and hardware that they produce. System design must include taking responsibility for the total system, comprising people, software, and hardware.

Understanding how the software and hardware will affect people in terms of how they could use it, what tasks it will and will not be good for, what changes it will require from users, and how it might improve, impair, or otherwise change work, are all issues that must be addressed in design and are fundamentally the responsibility of the system designer. Taking seriously the concerns of people who use or are otherwise affected by a computer system will require changes to the practices and methods of system design. Understanding users and their tasks is a central concern of the system designer. Scenarios provide explicit user and task information, which can better equip the designer to accommodate the rich perspective of the people and work for which he or she is designing computer systems.

“In developing the technology of computer systems it is often necessary to focus upon properties of the technology. However, in developing systems that are intended to be used by people in the varied contexts of their work, private, social, and leisure activities, the focus of design must be on the suitability of the designed artifact to support and complement human activity. Developing technology that serves the needs of people, rather than vice versa, requires a revised conception of computer system design. The activities of design must still allow systems to be well engineered such that they are reliable, efficient, and easily maintained; however, over and above this, the people who will use and be affected by the design in the context of its usage must be sharply in focus in the design process. Scenarios that include rich information about users and their tasks can bring such a focus to the design. Scenarios are snapshots of human activity from which the designer can gain understanding about users and tasks, and through which users can see how any design is likely to affect them and their work. Users and designers need to be able to communicate with each other to understand the domain, users, and tasks and the possible design-induced changes that may come about.”

The style of such writing (sociological and literary) surely forms part of the psychological barrier to the adoption of the techniques advocated.

R. J. Wieringa's more conventional textbook *Requirements Engineering: Frameworks for Understanding* (Wiley, 1996) mentions scenarios in only a few places and is almost dismissive of the approach, but still manages to be characteristically clear in its description:

“The client often finds it easy to describe *scenarios* that the SuD [System under Design] will have to participate in. These are a rich source of material for the construction of behavior specifications and, at a later stage, for the validation of specifications as well as for acceptance testing of the product.”

Richard Stevens' *Structured Requirements* course (© 1993-5 QSS/REL Ltd) takes a more positive attitude, placing scenarios firmly as the basic method of structuring user requirements (whereas system requirements are organised primarily as a hierarchy of the system's functions).

“Structure is critical for handling all complex elements in the whole lifecycle... The basic structure of the user requirements is an operational scenario... the sequence of results produced through time for the users.”

Stevens prescribes a procedure for obtaining a scenario:

- Start with the end goal;
- Derive the necessary results to get to that point;
- Keep the set hierarchical;
- Break large steps into smaller steps;
- Review informally at each level.

The idea is to take a goal, such as “get \$50 from teller machine”, and work back through the subgoals which lead up to it, such as “identify oneself to machine” and “specify amount wanted”. These steps can in turn be analysed further if necessary. The method then takes a twist:

“Some results... do not occur in a linear sequence. While there is a baseline [sequential] process, there will also be conditionality, repetition, and asynchronous aspects to be considered. ... [Such] factors are links between requirements, representing time or logical relationships.”

The idea that requirements may be expressed informally (in English or other natural languages) but linked with precision in a variety of ways seems to be unique. Users can effectively specify not only that a requirement normally follows another, but that certain requirements apply **WHENEVER** some condition occurs. The emphasis on sequence also makes necessary the possibility that some requirements are asynchronous, inherently parallel. For example, the engine and the gearbox of a car must both be assembled and tested before being mounted in the car's frame, but there is no implied sequence between the two processes. It is no accident that QSS' requirements engineering tool, DOORS, has been designed to organise requirements hierarchically with cross-links.

It seems a safe bet that we will hear much more about scenarios in requirements engineering as more organisations latch on to the power and economy of the scenario approach.

The Management of Change

More recently, Ed Yourdon writing in Byte's State of the Art pages, *When Good Enough is Best* (Byte, Sept 1996, 85-90), pointed up the huge problem of specifying and building systems in the face of constant change. A change in a requirement may have impact throughout a system; if that system is not even complete, the combined effect of multiple changes may be disruptive. It may be best to build systems rapidly and imperfectly (to meet an early specification) rather than to attempt to build a perfect system, only to have to modify it constantly before release:

“Our expectations for bug-free software developed on time, within budget, and with every feature that end users have asked for is unrealistic. By definition, these goals impose four interrelated constraints on developers, and by attempting perfection in one area, developers will likely create havoc in at least one other area...”

“That’s why just producing an up-front specification for good-enough software isn’t enough. You and your end-users must engage in a *dynamic* reevaluation of the specification, because most development projects take place in what author James Bach justly calls a “mad world”. On a daily basis, the project team may find that the hardware has changed, the tool vendors have changed (some bankrupt, others brand new), government regulations have changed, the economy has changed. Consequently the user’s requirements for the system change, and the design and implementation must change, too.”

Apart from the hinted blurring of the user/system requirements distinction, this is an admirably clear expression of the challenge of change. Yourdon suggests that “prototyping (i.e. rapid iterative development) is entirely compatible with mad-world, good-enough software development”, and argues convincingly for

“*requirements management*, an activity that comes before the detailed analysis and design modeling handled by tools... “

Yourdon identifies prioritisation as the key response to change. Not everything can be done in limited time with limited money and effort. Three questions to ask the customer, to evaluate priorities effectively, are:

- Which requirements are essential (i.e. without them, the system can’t be used at all)?
- Which requirements are important (i.e. without them, the system will be difficult or unpleasant to use)?

- Which requirements are optional (i.e. the “bells and whistles” that would make the system fun and interesting to use)?

In his view,

“...the tools used for this activity are word processors, because users typically describe their requirements in imperative English sentences like “the system must do X, Y, and Z”. Associated tools such as RTM, DOORS, and Requisite Pro can help describe the priority, cost, risk, and other attributes of each requirement... and help ensure that the team remains focused on the must-have features.”

This viewpoint takes in the need to give requirements attributes so as to prioritise them effectively, but seemingly ignores both the need to structure requirements into a comprehensible model (whether of the problem for users, or of the system for developers), and to link related requirements and constraints together. The implication that all systems should be developed in a prototyping style similarly cannot be taken absolutely literally. What all developers can surely agree with is the belief that close attention must be paid to user requirements: and since these can change at short notice, development must have a suitably short cycle time (or “wavelength”) to enable a continuing and effective requirements engineering dialogue.

Postscript

This review has given examples of quite diverse viewpoints on system development. Through all the differences, their authors share a common concern for accurate definition of systems. Understanding of how to achieve this has broadly increased in sophistication by moving away from the “soldering iron and wire-wrap gun” (Frederick Brooks) to the detached, problem-oriented definition of user requirements.

(c) Ian Alexander (iany@easynet.co.uk) 1997

CORE-Blimey!

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

DB or not DB – That is *not* the Question

In previous contributions I have noted that there is, for large system support an immense data handling problem due to the large number of people, locations, organisations, disciplines and cultures with legitimate interest in the products. The obvious candidates for assisting, and perhaps playing a central role are database systems, and there have been many attempts to use them. I have been involved with or observed such attempts for 20 years and each has failed to deliver the improvement that should be possible.

Here I shall try to explain why current database system technology is not sufficient, but why apparent rivals or

combinations with rivals also fail. First I discuss key features I think necessary for system support, then I provide a crude three-way classification of the types of support tool currently in use and why they fail singly or in combination on one or more of the required features.

Key features are: capacity, inter-relationship, distribution, autonomy, analytic power, flexibility and immediacy.

Capacity concerns the ability to handle a range of quantities of information – most critically, very large quantities of information. In order to facilitate support for some of the other features a coherent database approach of some kind is vital, but the nature of the data definition facilities provided strongly influences the viability of providing support for the range of data relevant to a computer system project.

Inter-relationship concerns the ability to support connections of various kinds in the (very large) information base. Here I am talking about more than just relationships between documents or references within documents. There are relationships between specific atoms of data in one document and one or more atoms in one or more other documents. It is often the ability to traverse these relationships which is the actual carrier of information, rather than the individual atoms alone. These relationships pass across disciplines and cultures, so they traverse technical and non-technical languages. It is not just a matter of supporting expression of inter-relationships. The means by which inter-relationship is supported can make achievement of other necessary features very difficult.

Distribution concerns the ability to support information across multiple disciplines, cultures, sites, organisations and media. In practical terms this can mean dealing with the existence of multiple copies (normalisation out of the window) and with variation and inconsistency (and out goes referential integrity) among the copies. This is partly to do with commercial and/or military security and partly to do with preventing single point of failure situations destroying the ability to make progress in development or update of project data.

Autonomy concerns the ability of team members to proceed independently unless and until a need for co-ordination arises. In practical terms it also concerns allowing team members to proceed if possible with minimum risk even if such co-ordination is needed but not provided promptly. This requires maximising access to information and co-ordinating concurrent use (particularly modification) with minimum delay and maximum chance of detecting clashes.

Analytic Power concerns the ability to perform a range of algorithms, many involving deep recursion. The algorithms may cross between disciplines and cultures – and thence notational conventions. It requires the ability to traverse atoms of data and chains of such atoms spread between different project concerns – treating chains as sets, lists, networks or any other relevant connecting criterion. It requires the ability to do almost any operation on the data from generating and running a simulation (if the data permits) though compilation or system assembly and traversing dependency chains, such as fault tree analysis or data flow or component usage.

Flexibility concerns the ability to change/extend information or information models or model rules or information presentation for specification – across all disciplines and cultures involved in the project. It requires support for addition of whole new concepts, linking them with existing concepts and addition of checking, deduction and translation algorithms. It requires addition of new reports, supported by new diagrammatic or textual or animated representation methods and linking them to existing reports. It requires support for the presence of inconsistency without losing a knowledge that it exists, or of its location. It requires support

for the ability to proceed on a “What if ...?” basis so that experiments can be tried and accepted or rejected.

Immediacy concerns the ability of individuals readily to enter or get at and understand the information they want, even if it is not already covered by a report generator or input form, without having to wait for an expert who understands a complex underlying data model to write a program which may or may not do what the individual really wanted. This is the feature that is the real killer when it comes to trying to solve all the problems of specification, while providing simultaneous support of the other features already discussed.

Current support environments use one or more of three basic types of tool: database tools, specialist language tools and bespoke method tools. The above range of features is very poorly supported in environments based on one or more of these tools categories of support tool. That is because they each fail in critical areas of required feature support and where they are combined it is in an incomplete or incompatible way. There is not space here for me fully to justify these comments, but I shall give a few reasons.

Database tools are primarily thought of as based on the relational model, though most mix relational capabilities with more general purpose programming capabilities via SQL and non-SQL language access to the data. In recent times the possibility of Object Oriented databases has been added, leading to the relational religion being extended to try to encompass this upstart technology.

Database systems in current use provide significant support for capacity, inter-relationship, distribution and autonomy. They are a disaster area for immediacy and provide support for flexibility of a sort, but at great cost in the area of immediacy. Their support for analytic power is pathetic unless the user descends to low level languages like C or COBOL – and then that is again at the expense of a large drop in support for immediacy.

Interestingly, database system approaches are intrinsically bad for some aspects of inter-relationship, even though one of their primary claims of strength is in this area. They are bad on two fronts: first, they make it difficult, without special action by their users or support team to find all there is to know about any given entity and second because the most efficient way to store inter-relationships leads to a great proliferation of tables containing basically unintelligible numbers or barely intelligible short mnemonics – leading to a great degradation in immediacy for most lay users.

Specialist language tools are a special categorisation I have invented for language support that embraces some database-like features within a general purpose programming capability. The two primary examples I have in mind are Prolog and Lisp.

These tools offer greater support for some aspects of immediacy and/or flexibility, but only at the expense of poor support for capacity and distribution. If you are working in a

large development environment – the sort where there are multiple, interacting but autonomous projects - they also damage support for inter-relationship, even when they sell themselves on the basis of their ability to handle inter-relationship. This is because they require the tool users to remember which component details are stored in which places.

Bespoke method tools are those written with the entire and only purpose of supporting a specific method of specification or development. These include formal methods tool sets, CASE tools and Programming Workbench types of environment (Integrated Development Environments).

These tools may appear to some to provide better support for autonomy, in that they either do not attempt to force co-ordination among tool users, or provide it in a way which does not require much interaction with things like project and configuration management – but it is as well to remember that the extreme implementation of autonomy is anarchy and most such approaches preserve autonomy at the expense of loss of important controls. They also fail at inter-relationship and serve immediacy only in a very limited way. For many potentially interested users of their products, immediacy is non-existent either because they can not deduce that the product concerns them or can not understand how and where it concerns them.

Attempts at combining tool types founder in one or more of several ways. First, they are not compatible – and in particular have non-existent or rudimentary interfaces. Second because they are treated as complementary, dealing with wholly separate concerns, when in fact those concerns are strongly interwoven. Each of these is an infringement of the inter-relationship requirement. Third they are each, in

their own way inflexible, being based on an almost religious determination to follow a given theory (e.g. relational or logic programming or data decomposition) which is not adequate alone and to which addition of extensions is very difficult.

One common result of trying combinations of these tool types is a severe over-control of project operations – destroying autonomy, even though component tools have the power to facilitate autonomy. This leads many to dispense with or resist such combinations – leading to the extreme of autonomy: anarchy, resulting in under-control, confusion, communication failure and ultimately project failure.

A combined approach is essential, but current attempts are impractical. We might stumble along with the current tools or combinations of tools and make up for the defects by making our industry more professional, thus raising the size of system we can build successfully using them, but the size of system we can get to on in that way is still small. Hence we must find ways to reduce the sub-optimal nature of tool support or stop building big systems or take out huge and hugely expensive insurance cover on each such system we start.

I do not believe we can stop people attempting to build these systems and I prefer not to be involved in trying to explain to people I have helped kill that it is alright because the insurance covers it, so the only option is to find ways to improve. In a later contribution I hope to show that there are ways to improve, but that they require a few computer religions to modify their gospels.

Geoff Mullery
geoff_mullery@dial.pipex.com

RE-Publications

Reviews of recently published books about requirements engineering.

Book Review: “Requirements Engineering A good practice guide” by Ian Sommerville & Pete Sawyer. John Wiley, 1997. ISBN 0-471-97444-7 paper

Reviewed by Ian Alexander

Sommerville has an excellent reputation for his research and his definitive textbook ‘Software Engineering’ (5th Edition, Addison-Wesley, 1996). It was therefore with some interest that I awaited ‘Requirements Engineering’.

The book is organised into the chapters that one might expect, from requirements elicitation through analysis, modelling, validation, and management to the ‘more detailed’ chapters on critical systems, structured methods, and formal specifications. Introductory chapters discuss process improvement and the requirements document.

But below that level, it is structured in an unusual way. All the chapters in the middle part are organised as ‘guidelines where we make practical suggestions for improving requirements engineering processes’. Each ‘guideline’ is a section (as 5.1, etc) whose title is a piece of advice, such as ‘Define System Boundaries’. These inevitably tend towards motherhood-and-apple-pie but are nevertheless good and useful concepts. In the title area there is then a box with the four subheadings ‘Key benefit’, ‘Costs of introduction’, ‘Costs of application’, and ‘Guideline type’ (basic, intermediate or advanced). The first paragraph then briefly expounds the guideline, and the rest of the section, generally a couple of pages, enumerates the benefits and the means of implementing the suggestion.

I use the word ‘enumerates’ advisedly, for the guidelines are set out almost entirely in text, unsupported by figures or tables. (There are a few in the final (advanced) chapters.) This seems surprising, given that requirements engineering is a practical subject, and some examples of well-written and well-laid out requirements complete with identifiers, section headings, and various types of attribute (priority, benefit,

cost, systems affected, ...) would be at least helpful if not essential. The heavy emphasis on text means that the book does not present the reader up-front with a process or task model of the subject: this is all the more surprising, given that the introductory chapters are closely focussed on processes. The authors undoubtedly have a model of the subject in their minds, but it leaks out rather than being stated explicitly.

The subdivision into some 66 short guidelines further fragments the account; there is not necessarily any particular connection between one guideline and the next. Chapter 2 does identify which guidelines are basic and which advanced, complete with a blank 'Usage' column for readers to fill in to suit their organization, but this approach would make the use of the book a lengthy process.

Several important aspects of requirements engineering are scarcely addressed: for example the nature of tool support, or the basic distinction between user and system requirements. On this last point, for example, the reader is exhorted (in just over one page) to make a business case for the system, 'as a separate section of the introduction to the requirements document'. Such a section could readily be overlooked, and a systems engineer might attempt to write one in a morning given the rather low importance the guideline seems to have. For example 'There are no significant costs involved in introducing or applying this guideline'. On the contrary, the elicitation and refinement of business requirements, to describe the problem to be solved, may be an intensive, controversial and time-consuming process involving several reviews and all levels of management.

The suspicion that this book is rather too narrowly IT-centric is reinforced by guideline 4.5 in particular: 'The operating environment of a system consists of the host computer, ...

and other hardware and software systems which will interact with the system'. Perhaps a bank's customers using its teller machines, or the patients being carried in ambulances, are essentially computer peripherals? The operating environment of a system such as a fighter aircraft (to take a contrasting example) is not only the avionics and airframe, but includes the pilot, the weather, and indeed the enemy. Requirements framed only in terms of hardware and software would fall seriously short of reality.

In addition, there is rather little emphasis on the crucial role that requirements play in business: to control and manage risk. Risk is indeed discussed, but again from the narrow perspective of assigning risk (attributes) to requirements: undoubtedly necessary but not sufficient. The guideline (5.8) in fact is somewhat dismissive: the type is stated to be 'advanced' rather than 'basic' (i.e. essential) and 'as risk assessment is not widely practised, the principal problem .. is the lack of people with risk assessment experience'.

The book lists rather few references, and these are not collected into a section at the end.

The book's best points are that it is very clearly and ably written, as one would expect, and that it is full of the accumulated common sense of some experienced system engineers. The guidelines are all useful and interesting, making many excellent suggestions. Experienced requirements engineers will want to look at a copy to see what Sommerville has to say; I'd be more hesitant in recommending it as a textbook for novices. Its approach is entirely different from R. J. Wieringa's book (Wiley, 1996) which is far more visual but also more academic. It is intriguing that Wiley have chosen to publish both texts: perhaps neither exactly fulfils the business requirement for such a book.

RE-Calls

Recent Calls for Papers

Call for Viewpoints

"Viewpoints" is a regular section in the Requirements Engineering Journal for airing reader's views on requirements engineering research and practice. Contributions that describe results, experiences, biases and research agendas in requirements engineering are particularly welcome. "Viewpoints" is an opportunity for presenting technical correspondence or subjective arguments. So, whether you are a student, teacher, researcher or practitioner, get on your soapbox today and let us know what's on your mind...

Please submit contributions electronically to Viewpoints Editor, Bashar Nuseibeh (ban@doc.ic.ac.uk). Contributions less than 2000 words in length are preferred [The REJ is published 4 times a year by Springer. Submissions are accepted at anytime and are subject to review]

Past columns include:

- Bashar Nuseibeh (Imperial College), "Conflicting Requirements: When the Customer is not always right!"
- Geoff Mullery (Systemic Methods), "The Perfect Requirement Myth"
- Laurence James (GEC-Marconi) "What's Wrong With Requirements Management Tools?"
- Richard Veryard (Texas Instruments), "Demanding Solutions"

Forthcoming columns include:

- Richard Stevens (QSS Ltd), "Last words on the (project) flight recorder: A plea for user requirements"
- Tom Gilb (Result Planning), "Towards the Engineering of Requirements"

See <http://www.mac.co.umist.ac.uk/RE/Journal.html> for more details of the Requirements Engineering Journal

Ninth IEEE International Workshop on Software Specification and Design (IWSSD9), Ise-shima, Japan, April 16-18, 1998

<http://salab-www.cs.titech.ac.jp/iwssd9.html>

In conjunction with International Conference on Software Engineering 1998. Sponsorship by the IEEE Computer Society and cooperation from ACM SIGSOFT have been requested.

Call for Contributions

The International Workshop on Software Specification and Design (IWSSD) is a leading international forum for research on software architecture, concurrent, distributed and real-time systems, formal models, and requirements and design methods. The workshop has succeeded in combining a widely cited and prestigious forum for publication with a programme based on working-group sessions and plenary sessions which provide an informal, yet focused, setting for discussions. Workshop attendance is by invitation only, based on a submitted paper.

As with previous meetings, IWSSD9 will use working-group discussions in order for participants to focus on their shared concerns in representing and reasoning about models of software-intensive systems. Following workshop tradition, this meeting's timing and location will give researchers the opportunity to complement a major professional meeting (ICSE '98) with a high-quality research workshop.

The program committee is requesting the submission of research papers outlining novel research directions, research projects or research proposals. The papers should make a case for the significance and originality of the work and

clearly describe (preliminary) results, infra-structure or exploratory studies on which the work is based. Of particular interest is the rationale underlying the work particularly if it identifies gaps in research and difficulties that have not been well understood. Papers should not exceed eight pages, including figures. Managerial, organisational and resource issues should not be included. Contributions are sought across the broad range of work in software specification and design.

Common case study material will be used to support submissions and to drive the programme. Prospective participants are strongly encouraged, though not strictly required, to use the case study in support of their submission. The principal case study is available at:

<http://salab-www.cs.titech.ac.jp/iwssd9.html>.

The workshop will be based around intensive working-group sessions organised in four separate tracks with plenary sessions to feedback results from working groups and for distinguished keynote speakers. The final selection of tracks will be determined by the submissions.

Proposed tracks, reflecting important themes in software systems engineering to which specification and design techniques are required to respond, are:

Flexibility, Mobility, and Extension

A track focussing on systems architecture with a particular emphasis on the development of systems in which the structure may change or in which components may be mobile.

Safety, Security, and Performance

A track focussing on analysis with a particular emphasis on the identification of, and reasoning about, important system properties.

Heterogeneity, Interoperability, and Legacy

A track focussing on the specification and design of heterogeneous, open and interoperable systems and on the systems in which integration with a legacy infra-structure is an important issue.

Traceability, Integrity, and Change

A track focussing on the traceability of requirements, specification and design information and the management of the integrity of system development documents in the face of change.

Important dates:

Submission deadline: November 14 1997

Acceptance notification: January 12th 1998

Final copy due: January 26th 1998

The first page of the paper should include its title, names and affiliations of the authors, and a designated contact author with his or her address, phone and fax numbers, and e-mail address. To assist the programme committee you are asked

RE-Bites...

A certain company, well known for its user-friendly software, was developing a new version of its operating system. One of the requirements laid down by the systems engineer was that the system should respond to any user command within two seconds. The development was going very well, except that one particular operation, which required some intensive disk access, was taking rather longer than two seconds to complete. A team was formed to address the problem, and spent two weeks of intense effort trying to optimise this operation.

Eventually, the team concluded that it was not possible to meet the requirement, and went back to the systems engineer to ask that the requirement be relaxed. The systems engineer patiently pointed out to them that the requirement was not that the operation be completed, merely that the system should *respond* within two seconds. All they needed to do to meet the requirement was to display a message acknowledging the user's command, and perhaps giving an estimate of how long the operation would take...

to indicate your preferred workshop track. Five copies of the paper must be received by November 14th at the following address:

Prof. Ugo Buy
EECS Dept. (M/C 154)
University of Illinois
851 South Morgan Street
Chicago, IL 60607
USA
TEL: (312) 413-2296 (Direct Dial)
FAX: (312) 413-0024
EMAIL: buy@figaro.eecs.uic.edu

In addition the paper title, abstract, names and affiliations of the authors, and a designated contact author with his or her address, phone and fax numbers, and e-mail address should be sent electronically to:

Prof. Anthony Finkelstein Email: acwf@cs.city.ac.uk

Papers may also be submitted electronically, details of how to do this are given on the workshop web page:

<http://salab-www.cs.titech.ac.jp/iwssd9.html>

Accepted papers will appear in the workshop proceedings, to be published by the IEEE Computer Society Press. Every accepted paper should be presented at the workshop by one of its authors.

General Chair:

Dr. Motoshi Saeki
Tokyo Institute of Technology
Department of Computer Science
Ookayama 2-12-1, Meguro-ku
Tokyo 152, Japan
TEL: 81 3 5734 2192 (Direct Dial)
FAX: 81 3 5734 2911
EMAIL: saeki@cs.titech.ac.jp
URL: <http://salab-www.cs.titech.ac.jp>

Finance Chair

Mr. Hisayuki Horai
Fujitsu Laboratories Ltd.
Information Science Laboratory
Nakase 1-9-3, Mihama-ku
Chiba 261, Japan
TEL: 81 43 299 3585
FAX: 81 43 299 3075
EMAIL: horai@iias.flab.fujitsu.co.jp

Programme Chairs

Prof. Ugo Buy
EECS Dept. (M/C 154)
University of Illinois
851 South Morgan Street
Chicago, IL 60607
USA
TEL: (312) 413-2296 (Direct Dial)
FAX: (312) 413-0024

EMAIL: buy@figaro.eecs.uic.edu

Prof. Anthony Finkelstein
City University
Department of Computer Science
Northampton Square
London EC1V 0HB
UK
TEL: 44 171 477 8556 (Direct Dial)
FAX: 44 171 477 8587
EMAIL: acwf@cs.city.ac.uk
URL: <http://www.cs.city.ac.uk/finger/acwf>

Programme Committee

Tsuneo Ajisaka - WU, JA
Shing-Chi Cheung - PRC, HKUST
Eric Dubois - UN, BE
Steve Easterbrook - NASA-IV&V, USA
Jose Fiadeiro - UL, PT
Carlo Ghezzi - PM, IT
Patrice Godefroid - Bell Labs, USA
Michael Goedicke - UEs, DE
Sol Greenspan - GTE, USA
Michael Harrison - UoY, UK
Connie Heitmeyer - NRL, USA
Shinichi Honiden - Tosh, JA
Hisayuki Horai - Fuj, JA
Paola Inverardi - CNR, It
Daniel Jackson, CMU, USA
Takuya Katayama (JAIST)
Gerald Karam - ATT, USA
Jeff Kramer - IC, UK
Tom Maibaum - IC, UK
Shin Nakajima - NEC, JA
Bashar Nuseibeh - IC, UK
Debra Richardson - UCI, USA
Catalin Roman - G. Wash., USA
Kevin Ryan - UoL, IRE
Wilhelm Schaeffer - UP, DE
Kenji Takahashi - NTT, JA
Lincoln Wallen - UofOx, UK
Roel Wieringa - VUA, NL
Jack Wileden - UMASS, USA
Alex Wolf - UC(B), USA
Jim Woodcock - UofOx, UK

5th International Eurographics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'98, Cosener's House, Abingdon, U.K., June 3-5, 1998

Theme

The workshop will provide a forum for the exchange of ideas on diverse approaches to the design of interactive systems. The particular focus of this year's event is on models (e.g., of devices, users, tasks, etc.) and their role in supporting the design and development of interactive systems. As in previous years we maintain our interest in the

use of formal representations and their role in supporting the design, specification, verification, validation and evaluation of interactive systems. Contributions pertaining to less formal representations of interactive system designs and model-based design approaches are also encouraged. The workshop aims to encourage an exchange of ideas between these different research fields.

Suggested Topics

Model-based design of interactive systems
 Task-based design of interactive systems
 Formal description of user related properties
 Formalisms for the specification of interactive systems
 User interface architectures
 User interface development environments
 Validation of formally specified properties of interactive systems
 The role of representations (formal/informal) in the design of interactive systems
 Specification-based evaluation of usability
 Cognitive models in interactive systems design
 Empirical assessment of formal design, specification and verification approaches

Submission Categories

Full papers addressing in-depth questions on the above topics (up to 15 pages).
Experience papers reporting on the practical application of such techniques or industrial perspectives on the above topics (up to 5 pages).
Position papers. In a few cases participation will be possible if a position paper is submitted with your views on the workshop topics (up to 2 pages).
Demonstrations of tools to support the design of interactive systems. A description of the demonstration (up to 2 pages) should be accompanied by a video (up to 5 minutes).
Video presentations of tools will be accepted as well. A description of the demonstration (up to 2 pages) should be submitted with the video (2 to 8 minutes).

Format of submissions

All submissions should be accompanied by a cover sheet indicating: submission category, title, name and contact details of author(s), keywords and a 100-word abstract.

Papers of all categories should follow the Springer Verlag format for Eurographics publications (see <http://www.springer.atnet.at/instructions/Eurographics.html>)

Video submissions should be in PAL/NTSC format.

Schedule

Expression of interest by January 9, 1998. Send an email with topic and author details to the address below.
 Submission of papers, videos and demonstrations by February 6, 1998. Post 4 copies of the paper and (where

applicable) one copy of the videotape to the address below.

Notification of acceptance by April 10, 1998.

Publication

All accepted papers will be published in the workshop informal proceedings, which will be printed and distributed prior to the workshop. In addition, the review committee will select the most relevant papers to be published in the Focus on Computer Graphics Series published by Springer-Verlag, along with the workshop reports.

Contact Address

Panos Markopoulos
 Department of Computer Science
 Queen Mary and Westfield College
 University of London
 Mile End Road
 London E1 4NS, UK
 tel. +44-(0)171-975 5257
 fax. +44-(0)181 980 6533
 email: markop@dcs.qmw.ac.uk
 URL: <http://www.dcs.qmw.ac.uk/research/hci/dsvi98>

Organising Committee

D. Duce (local organisation), Rutherford Appleton Laboratory UK
 P. Johnson (co- chair), QMW, Univ. of London, UK
 M. Harrison, Univ. of York, UK
 P. Markopoulos (co- chair), QMW, Univ. of London, UK
 J.C. Torres, Univ. of Granada, Spain

Paper Review Committee

M. Apperley, Waikato, New Zealand
 A. Dix, Univ. of Staffordshire, UK
 G. Faconti, CNUCE-CNR, Italy
 M. Green, Univ. of Alberta, Canada
 C. Johnson, Univ. of Glasgow, UK
 P. Palanque, LIS-University of Toulouse, France
 A. Puerta, Stanford, USA
 S. Schreiber, Siemens, Munich, Germany
 N. Sukaviriya, IBM Thomas Watson Research Centre, USA
 J. Vanderdonckt, FUNDP, Namur, Belgium
 J. Coutaz, CLIPS-HCI, Grenoble, France
 D. Duke, Univ. of York, UK
 P. Gray, Univ. of Glasgow, UK
 R. Jacob, Tufts Univ., USA
 D. Olsen, Carnegie Mellon, USA
 P. Paternó, CNUCE-CNR, Italy
 G. Reynolds, CSIRO, Canberra, Australia
 C. Stephanidis, ICS-Forth, Greece
 P. Szekely, Univ. of Southern California, USA
 M. Wilson, RAL, UK

Sponsors

Eurographics Association, ERCIM

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://research.ivv.nasa.gov/~steve/resg/>

Web Pages

The BCS RESG home page can be found at:

<http://porta.cs.york.ac.uk/bcs/resg/>

Back issues of Requirenautics Quarterly:

<http://research.ivv.nasa.gov/~steve/resg/>

A detailed description of UML can be found at the Rational web site:

<http://www.rational.com>

Books

Graham, Ian "Object Oriented Methods". Addison Wesley (ISBN 0-201-59371-8)

Mailing lists

Requirements Engineering Newsletter

A Requirements Engineering Newsletter is published as an educational service by Prof. Anthony Finkelstein at City University. If you wish to contribute send your material to the moderator at: requirements@cs.city.ac.uk

Subscription (or removal) requests should be sent to: requirements-request@cs.city.ac.uk Send an email containing

subscribe <address>

or

unsubscribe <address>

The Requirements Engineering Newsletter is archived at:

<http://web.cs.city.ac.uk/homes/acwf/rehome.html>

or <ftp://ftp.cs.city.ac.uk/pub/requirements/>

Software Requirements Engineering Mailing List

To subscribe to the Software Requirements Engineering (SRE) mailing list, e-mail listproc@jrcase.mq.edu.au, with the only line in the body of the message:

subscribe SRE your-first-name your-second-name

Articles to the SRE mailing list should be sent to SRE@jrcase.mq.edu.au.

Tools

The NASA/WVU Software Research Lab has now released TCMJava, a Java port of the original TCM (Toolkit for Conceptual Modeling), developed at the Vrije Universiteit Amsterdam. TCMJava provides a collection of graphical editors for editing software specifications. Editors are available for

Graphs: generic graph diagrams, entity-relationship diagrams, class-relationship diagrams, state transition diagrams, recursive process graphs, data (and control) flow diagrams, and JSD process structure and system network diagrams.

Tables: generic tables, transaction decomposition tables, transaction-use tables and function entity type tables.

Trees: generic textual trees and function decomposition trees.

TCMJava is free, and runs on any machine (as long as there is a Java virtual machine available). It can be obtained from:

<http://research.ivv.nasa.gov/projects/WHERE/TCMJava/>

RE-Actors

The committee of RESG

Chair: Dr. Bashar Nuseibeh, Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ. 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. N.A.M.Maiden@city.ac.uk, Tel: 0171-477-8412, Fax: 0171-477-8859

Secretary: Michael Bearne, Philips Research Labs, Cross Oak Lane, Redhill, Surrey RH1 5HA. bearne@prl.research.philips.com, Tel: +44 (0)1293 815428, Fax: +44 (0)1293 815500

Membership Secretary: Dr. Sara Jones, School of Information Sciences, University of Hertfordshire, Hatfield, AL10 9AB. S.Jones@herts.ac.uk, Tel: 01707 284370, Fax: 01707 284303

Industrial Liaison Officer: Dr. Orlena Gotel, Systems and Software Engineering Centre, Defence and Evaluation Research Agency, St. Andrews Road, Malvern, Worcestershire, WR14 3PS. olly@hydra.dra.hmg.gb, Tel: 01684 894674

Publicity Officer: Dr. Andrew Vickers, Department of Computer Science, University of York, Heslington, York YO1 5DD, andyv@minster.york.ac.uk, Tel: 01904 434727, Fax: 01904 432708.

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

Newsletter Editor: Dr. Steve Easterbrook, NASA/WVU Software IV&V Facility, 100 University Drive, Fairmont, WV 26554, USA. steve@atlantis.ivv.nasa.gov, Tel: +1 (304) 367-8352, Fax: +1 (304) 367-8211

Newsletter Associate Editor (Features and Book Reviews): Dr George Spanoudakis, City University, Department of Computer Science, Northampton Square, London EC1V OHB. gespan@cs.city.ac.uk, Tel: 0171 477 8000 ext. 3701, Fax: 0171 477 8587.

Newsletter Associate Editor (Features and Event Reviews): Ian Alexander, 2 Dornton Road, London SW12 9ND. iany@easynet.co.uk. Tel: 0181 675 6915

RE-Creations

How to contribute to RQ

Please send contributions to Steve Easterbrook (steve@atlantis.ivv.nasa.gov) before the publication deadline. Submissions must be electronic copy, preferably plain ASCII text. A list of the kinds of contributions we welcome can be found in the January 1996 newsletter, or on the web at:

<http://research.ivv.nasa.gov/~steve/resg/rq5/ReCreations5.html>

Copy deadline

Issue 13 (January)	19th December 1997
Issue 14 (April)	27th March 1998
Issue 15 (July)	26th June 1998