Requirements Engineering Specialist Group

Requirements Quarterly

© 2011 RESG www.resg.org.uk RQ57 (September 2011)

Contents

RE-soundings

From the Editor

Ten years ago, give or take a month or two, then-editor Pete Sawyer began his introduction to the summer 2001 issue of *RQ* with this observation: "In many ways RE practice changes only slowly; that's how it should be for an activity so critical to the development process." (*RQ* 24, July 2001) And I certainly wouldn't disagree. Core tenets of the discipline go unchanged for obvious reasons. But what does change of course, and very rapidly, is the environment in which RE is practised.

Ian Alexander notes the shifting scene of the last decade in his final message as RESG Chairman. I'm sure you'll all join me here (and, indeed, in person at our 7 September AGM!) in thanking him for his inspiring dedication and for a job well done. I for one hope he'll long continue to grace these pages with the fruits of his experience and expertise.

More and more development projects are now carried out by distributed teams collaborating across time-zones and hemispheres. Guest articles by Soo Ling Lim and Camilo Fitzgerald highlight two different aspects of this globally connected development space: Soo Ling presents a novel way to manage stakeholder discovery and prioritisation through social networks and Camilo shares lessons learned in Japan about upfront requirements analysis.

Shifting trends in the way projects are managed are also always of great interest to the requirements engineer, as two recent events on contrasting methodologies have again made clear. The First International Workshop on Agile Requirements Engineering threw up as many opportunites as it did challenges for projects that aim for agile delivery. Meanwhile, the i* Showcase demonstrated how an established framework remains a powerful tool in many large commercial projects and, further, how it still benefits from the results of fresh research. You can read extensive reports on both events in the following pages.

Finally, in case you were starting to feel smug about the progress we've made since Pete's words in 2001, we have a guest article from Tom Gilb and Lindsey Brodie that poses the fundamental question *What Went Wrong?*

I hope you enjoy the issue.

William Heaven, Editor

Chairman's Message

Well, the time has come around already for me to write my last Chairman's Message. The RESG is about the same age as the World-Wide Web. That is not a long time in the grand scheme of things, but quite an age in the world of software. Perhaps you remember when the UML was born, as a daring fusion of disparate modelling languages; or when Extreme Programming, now sedately named Agile, was the enfant terrible of methodologies. The world of software companies, too, has changed out of recognition, as the giants of the business have acquired one small company after another - and most of the big-name requirements tools, too.

We can't today imagine how the world of software development and RE will look in another couple of decades: but surely, very different from today. It seems a safe bet that the old world of lists of traditional requirements typed on to paper documents will be receding rapidly into history, as both writing and reading move further onto tablets and pads, and business settles in to life with the World-Wide Web as a given.

Already, online businesses are discovering requirements by analysing huge volumes of data recorded directly from customer behaviour - what sort of books and music I like to buy, how often, how much I spend, to give a familiar example.

There will surely be many similar innovations in the discovery of business needs and opportunities, and in the speed and ability of businesses to respond to them.

In all this time, the RESG has tried to respond to the very different needs of people in academia, industry and commerce by organizing meetings formal and informal; by publishing its newsletter, RQ; and through its website. Amongst all the discussion groups and mailing lists, there still seems a place for actual face-to-face meetings, workshops, debates, pub evenings, tutorials and the rest. I hope that the RESG will continue to provide food for thought, instruction, and opportunities for new ideas to flourish. Long live the RESG.

Ian Alexander, RESG Chair

RE-treats

Annual General Meeting & Summer Party

6pm, 7th September 2011 – Imperial College London

Venue: Level 4 Common Room, Huxley Building, 180 Queen's Gate, SW7 2AZ

RESG co-founder **Neil Maiden** will be giving a talk entitled Requirements Research and Practice: Stories Past and Future at this year's AGM.

Talk Abstract

Stories in different guises are one of the most effective requirements techniques. Use cases are a cornerstone of the UML. Scenarios are effective for discovering, validating and communicating require-ments. Stories and storyboards are an important design thinking technique. In this seminar the speaker will deliberately and outrageously overuse the keyword in the seminar title to describe previous uses of scenarios and stories to discover future system requirements, look back and tell stories about the early days of the RESG, and encourage collaborative storytelling as the future of requirements engineering.

Neil's Bio

Neil Maiden is Professor of Systems Engineering, Head of the Centre for Human-Computer Interaction Design and academic lead of the Centre for Creativity in Professional Practice at City University London. He received a PhD in Computer Science in 1992. He is and has been a principal and co-investigator on numerous EPSRC- and EU-funded research projects including SIMP, CREWS, BANKSEC, SeCSE, APOSDLE, TRACEBACK, S-Cube, MIRROR and CHOReOS, with a total value of £2.5million. His research interests include establishing the requirements for complex socio-technical systems, scenario-based design, service-oriented systems and cloud computing, goal modelling, and technology-enhanced creative design. He also investigates new ways of delivering results from these research areas into practice. Neil has published over 150 peer-reviewed papers in academic journals, conferences and workshops proceedings. He was Program Chair for the 12th IEEE International Conference on Requirements Engineering in Kyoto in 2004. He is on the Editorial Boards of the IEEE Software and Requirements Engineering Journal and a former member of the Editorial Board for IEEE Transactions of Software Engineering. He is also the Editor of IEEE Software's Requirements column. His details are available here.

The event will also feature a shorter presentation from **SanazYeganefard** from the University of Southampton about her work.

Free food and drinks will be served.

Registration is free. To register, please email Dalal Alrajeh at dalal.alrajeh04@imperial.ac.uk.

Non-RESG Event

Mastering the Requirements Process

17-19 October 2011 / 20-22 February 2012 – London

James Robertson, Atlantic Systems Guild

Requirements are the most misunderstood part of systems development, and yet the most crucial. Requirements must be correct if the rest of the development effort is to succeed. This workshop presents a complete process for eliciting the real requirements, testing them for correctness, and recording them clearly, comprehensibly and unambiguously.

Delegates will learn to:

- Determine your client's needs-exactly
- Discover the real business, and how to improve it
- Write requirements that are complete, traceable, and testable
- Precisely define the scope of the project
- Discover all the stakeholders-and keep them involved
- Get the right requirements quickly, and incrementally
- Learn state of the art requirements techniques
- RESG members get a 10 per cent discount

Visit www.irmuk.co.uk/1/ for full seminar details or contact IRM UK on +44 (0)20 8866 8366 or email customerservice@irmuk.co.uk.

Non-RESG Event

Mastering Business Analysis

3-4 November 2011 / 19-20 April 2012 – London

James Archer

Business analysis provides the foundation for almost every kind of business change. The craft of business analysis is to investigate the business, to find its problem hot spots and recommend ways to improve them. This two-day seminar and workshop in business analysis gives you the skills and tools to discover your client's real business, and to determine and demonstrate the best ways of improving it.

Delegates will learn to:

- Discover real business needs, not just the most talked-about ones.
- Improve the business processes by applying automation or other means.
- Define the most beneficial scope for the analysis project.
- Use models to understand and communicate the business processes, and ensure stakeholders also understand.
- Understand how to employ business events as a way of partitioning the business for easier understanding.
- Be better at interpersonal communication.
- Think systemically, and find truly the best way to improve your client's business.
- Be a better business analyst

RESG members get a 10% discount.

Visit www.irmuk.co.uk/1/ for full seminar details or contact IRM UK on +44 (0)20 8866 8366 or email customerservice@irmuk.co.uk.

Careers in RE

November 2011 - University of Westminster

Further details will be published in the next issue. In the meantime check for updates on our website, or read the report of 2010's event in the following pages.

Traceability Event

January 2012 – BCS Southampton Street, London

Details to follow next issue.

Post-Doc Early Researcher Workshop

February 2012 – Imperial College London

Details to follow next issue.

In the meantime, have a look at the format of this year's successful event:

www.resg.org.uk/index.php/Post_Docs_2011

More?

If you have an idea for an event – especially one based outside of London – please get in contact with a member of the committee (see the back page).

RE-member

Where Do Software Requirements Come From?

Joint event with BCS East Anglia

7.30pm, 12 April 2011 – Cambridge

Venue: Red Gate Software Ltd., Newnham House, Cambridge Business Park, Cambridge, CB4 0WZ

This 2-hour event was organised by **Charles Looker**, Secretary of BCS East Anglia, who invited representatives of the RESG to talk to his branch on the challenges and practice of requirements capture. Three members of the RESG committee – Ljerka Beus-Dukic, Alistair Mavin, and Dalal Alrajeh – took up the invitation with the plan to present three different views on methods for capturing requirements: academic, research-oriented and industrial. Unfortunately, last minute committments prevented Alistair Mavin from attending on the day. But the academic and research-oriented viewpoints were given a good airing.

Ljerka Beus-Dukic spoke from an academic perspective. She talked about her experience in teaching the RE course for undergraduates. She discussed the practical challenges that often students find when faced for the first time with an RE problem and how she helps them to think and grasp stakeholder's requirements.

Dalal Alrajeh spoke from a research perspective. She discussed the problems that face requirements engineers particularly with the rapid development of new technologies, and talked about how techniques from artificial intelligence and machine learning can provide specific automated support for capturing requirements.

The enthusiastic audience of about 15 were very much engaged with questions during and after the talks.

Thanks to BCS East Anglia!

Dalal Alrajeh

i* / iStar Showcase

21 June 2011 – City University, London

Ian Alexander welcomed everyone to London and to the RESG. The purpose of the RESG was sharing and dissemination of RE knowledge and techniques. Goal modelling was still seen as novel – or was simply unheard of – in industry. Engineering practice should be soundly based on proven engineering theory; conversely, researchers had to focus on results that would be useful to industry.

Eric Yu (University of Toronto), "the father of i*", introduced the workshop by explaining the purpose of i*. It addresses the 'early' part of RE. The basic concepts of i* are the 'strategic' actors and their goals. These actors are unlike the stick figures of UML in that they are imagined to have purposes, beliefs, and commitments which are vital to system development. Hence, discovering and modelling those purposes is critical.

The i* strategic dependency relationship grows out of this: one actor's needs are seen to be met, or are planned to be met, by the actions of another actor.

The i* strategic rationale model goes further, exploring how each actor can go about achieving their and other actors' goals. This can be by actual tasks (representing functional needs) and softgoals (representing qualities of service, and hence non-functional needs, i.e. qualities and constraints). Relationships include means-ends links (typically a task or sub-goal means to a goal end) and links for task-decomposition (into the four i* elements: task, resource, goal and soft goal). It is possible to show multiple alternative means-end approaches for a given goal (you can boldly label these with e.g. a diamond to indicate the alternatives). The SR model also permits softgoal dependency links, which are carried down from the (top-level) SD model of dependencies (hard and soft) between different actors, which may describe any combination of functional and quality needs.

i* does not attempt analysis at code or execution level (UML is now the standard way of doing that); nor does it represent time and temporal relationships in any way (you need some form of scenario analysis for that).

Goals, or course, may not be fully satisfied, or may be denied wholly or partially. i* permits ticks (checkmarks) and crosses to be placed on the links to show this.

Many materials including tutorials, guidelines and research papers are provided for free on the i* home page: www.cs.toronto.edu/km/istar.

Numerous, mainly academically-developed, i* tools are listed on a wiki: istar.rwth-aachen.de.

Jaelson Castro (UFPE, Brazil) took the chair for the workshop. There were first three relatively detailed talks of 15 minutes each.

James Lockerbie (City University) spoke on using i* to bridge air traffic management operational concepts (in text and graphics) and formal safety analysis and simulation (e.g. with Petri nets). i* modelling was combined with video-conferencing. People in air traffic control are well accustomed to building and interpreting complex models, so they had little trouble getting used to i* - the basic idea took them "five minutes", and then they needed half an hour to get to grips with individual models and the concepts used. The models were best when strategic: people specially liked the "one page" treatment. People felt it was an effective tool for presenting scenarios – you would need to mark up i* models to do this properly.

Jorgen Engmann (formerly of City University) spoke on using i* and satisfaction arguments to model evolving requirements in public health. The mass of documentation – a reverse engineering challenge! - was analysed into simple responsibility tables with actors, their responsibilities (as i* SR models), conditions (in the environment), and reasons (as satisfaction arguments, themselves analysed as i* SR models). Effectively, the goals are representing the assumptions (steps in the reasoning for an argument). (A goal is something that an actor wishes to achieve; an assumption is something that must be achieved for an argument to hold – not the same thing, but a fair match.) The models could then be used for impact analysis. Making the models continually revealed new domain properties (candidate requirements).

Maria Carmela Annosi spoke on helping an organisation (Ericsson, in Italy) to transition to agile practices (Scrum) using i* strategic modelling and empirical knowledge. People with different responsibilities within the organisation were interviewed to understand their role, goals and dependencies on other people. The understanding was documented in i* SR models. Numerous kinds of analysis – such as of strategic contributions and of trade-offs – were then carried out on the i* models. This incidentally demonstrated the power of i*, as well as serving the organisation effectively. Unlike conventional approaches, i* assisted by aligning (possibly conflicting) goals. There was however resistance within the organisation to the adoption of agile practices.

The remainder of the workshop consisted of sixteen very short (3-minute!) presentations given by participants, each supported by an interesting poster at the back of the room for everyone to look at in the breaks and to discuss at the end of the afternoon.

Jaelson Castro himself kicked things off with his poster reporting on using i* in civil engineering – it was beneficial in a complex domain; engineers found it a steep learning curve; scalability was an issue.

Dominik Schmitz (RWT/Aachen University)'s poster was on managing requirements knowledge in control systems. Engineers were able to develop and read i* diagrams. They were surprised there was information in the goal models that wasn't in their functional block diagrams. Domain models (for specific control system domains) were helpful in getting new projects going. Such higher level RE models were useful to projects.

Fabiano Dalpiaz (University of Trento)'s poster was on applying Tropos to the Trentino region's TasLab initiative. i* provided the traceability between goals collected in interviews with service design (shown as tasks).

Jennifer Horkoff (University of Toronto) presented no less than four posters developed by people in her department and the University of Ottawa.

The first used i* very simply to model as-is and to-be alternatives. The key was to avoid modelling everything! Modelling was good for analysts and clients, and it brought out conflicting viewpoints clearly.

The second one used URN (including GRL, an i* variant) and (CMM-like) key performance indicators for small business performance management. A free open-source tool propagates results through the goal network.

The third looked at proactive adverse event (like a patient being given a wrong dosage) management in healthcare. GRL was used to model patterns of adverse events, together with scenarios modelled as small processes. Again, results were propagated through the models quantitatively.

The fourth poster described collaborative social modelling, with the purpose of designing a healthcare system. Stakeholders could not work with technical models, and a year went by. The early RE communication issues were then well resolved with i* SD and SR models. The stakeholders were much better involved, and it was not hard to translate the i* to outline design models. The full i* syntax was too complex, so a simplified syntax was used.

Eric Yu presented a poster by some of his colleagues on privacy goals with respect to systems that come with small print and lots of tricky settings on some Settings screen. Users want quality service (which may demand access to private information), but also privacy (even in emergency). There are clearly both dependencies and conflicts here. Low-level technical features contribute to the privacy goals; these are mediated by other goals normally only known to privacy / security specialists.

Xavier Franch (University of Catalunya) presented a poster on architecting hybrid systems – both social and technical. The team trained the stakeholders in i* (3 hours' training, then a week's practice, then a consolidation meeting). A method was developed to translate from stakeholders' goals to design solutions (architectures). The stakeholders (rightly?) gave biased viewpoints – their own points of view. The architectural models had to be rearranged and decomposed into workable designs.

Juan Trujillo (University of Alicante)'s poster was on i* for modelling requirements for a data warehouse.

Daniel Gross (University of Toronto)'s poster was on understanding stakeholder viewpoints in enterprise service-oriented architecture. Decisions are based on reasons for and against. An i* model was made showing the arguments for each option. A higher-level model showed the viewpoints of the stakeholders responsible for making the decisions. This was a very pure application of i*'s different types of link.

Anna Perini (University of Trento)'s poster was on regulatory compliance of requirements for healthcare systems. i* was used to model the relationship between the provisions of the law (29 articles of Italian law on privacy) and the activities needed to comply with the law, as well as the audit activities needed to demonstrate compliance.

The RESG would like to thank the staff of City University for organising the event and for doing all the legwork on the day!

© Ian Alexander 2011

First International Workshop on Agile Requirements Engineering: Workshop Report

The first international workshop on Agile Requirements Engineering was held on 26 July 2011 at the European Conference on Object Oriented Programming (ECOOP) at Lancaster, UK.

Introduction

Ever since agile first emerged in the 1990s, there has been debate about what role, if any, requirements engineering (RE) can play for agile practitioners and their customers. That agile development needs requirements is not disputed, but the relevance of the assumptions, methodologies, techniques and tools that make up the discipline that has become known as RE, is. Thus, while agile emphasizes incremental discovery and satisfaction cycles with face-to-face interaction rather than documentation, RE has traditionally stressed full understanding of requirements before commitment to coding and rigorously maintained, version-managed and traced requirements documents. Yet both of these views are stereotypes, rendered even less valid by the evolution that has occurred in both the agile and RE worlds. Thus, for example, techniques have emerged from the RE community for dealing with volatile domains where the requirements can't be fully known before coding begins; sometimes not even before deployment. Similarly techniques have been developed in the agile community for modelling, structuring, and analyzing requirements knowledge.

The aim of the Agile RE workshop was to take stock of the two worldviews to discover whether agile needs RE, and whether novel RE practices can deliver what agile needs. The workshop invited submissions of short papers describing experiences, challenges, vision, and ideas on the need for, or use of requirements engineering techniques in agile software development projects.

Keynote: "Agile projects are requirements-centric"

The workshop started with a keynote talk by **Wolfgang Emmerich** entitled "Working with User Stories" during which he presented his company's experience of working with user stories in Agile projects. Wolfgang is a professor of Distributed Software Engineering at University College London and Chairman of Zühlke Engineering, a medium-sized company that has been an early adopter of Agile Development and successfully completed more than 50 projects using the Scrum methodology.

A user story is a way of capturing requirements through simple concrete statements that are not technical in nature. They are described using the Role-Action-Goal format, for example: "As a spectator (role), I want a daily time table of matches at Wimbledon (action), so that I can attend the matches that interest me (goal)". User stories are understandable by all project stakeholders. They are used in testing, estimation, prioritization, planning, and in fact everywhere conversations about the project take place. User stories are recorded on index cards with annotations on the front and back of the card to record the story priority, status, estimate, and acceptance criteria. Details of the story are worked out in discussions between developers, users, analysts, and testers. A critical part of working out these details is to record acceptance test for the user story that will allow everyone to determine when the story is complete. But how do we know if the user stories are actually good? To assess the quality of user stories, Zühlke relies on the INVEST test devised by Bill Wake: a user story should be Independent, Negotiable, Valuable, Estimable, Small, and Testable. This test allows notably detecting "tip of the iceberg" requirements – simple statements of requirements that look innocuous but that actually hide a huge amount of work and complexities.

Users stories are typically gathered by a combination of user observation, workshops, and interviews. Stakeholders will often first express user stories that are too large to pass the INVEST test. A lightweight modelling technique known as story mapping is then used to capture these large user stories, called epics, and decompose them into activities and smaller stories. Such story maps help everyone understand the end-to-end use of the system. The horizontal axis of the story map is used to order the user stories chronologically (in the order you would refer to them to explain what people do with the system), the vertical axis is used to order the user stories by importance with the most important user stories at the top and less important ones at the bottom.

The story map provides the basis for defining the software architecture. As soon as the story map is developed, it is used to build a "walking skeleton" – a small application that supports the least number of necessary tasks across the full span of functionality. Building the walking skeleton is an important early step that allows developers to gain confidence that the architecture will work in practice. The backbone of the application is the abstract list of essential activities the application supports.

User stories are then used as the basis for estimating development effort. Effort estimation at the start of a project is notoriously difficult due to the large amount of uncertainties at this stage. Because such uncertainties decrease as development progresses, the idea is to try to postpone the need for estimation for as long as possible. Zühlke relies on two consensus-based techniques for estimating user stories development effort: Planning Poker which is quick and easy and good for most cases, and Wideband Delphi which is more thorough and detailed and good for more high-impact cases or when consensus is harder to obtain. These techniques are based on the "wisdom of the crowd", an idea that was illustrated by making us collectively estimate the weight of a motorcycle (the mean of our individual estimates arrived impressively close to the correct weight).

Wolfgang then presented how user stories are used to plan the development iterations and track progress. User stories have a very simple lifecycle; from being included in the backlog of stories for the next iteration, to being started, ready for sign off, and done. By displaying user stories on a "project wall", the status and progress of all users stories is clearly made visible to all at all time.

An essential part of agile development is the relation between user stories and testing. The back of a user story index card documents a set of acceptance criteria that provide a way for judging when a user story development is complete, and each acceptance criteria is linked to concrete acceptance test. Elaboration of the acceptance criteria and acceptance test is the point at which the detailed requirements of a user story are worked out. Agile development practice insists that the acceptance criteria must be written before the user story can be accepted for development. Wolfgang mentioned an anecdote where a user story was about to be included in the next iteration but was still missing its acceptance criteria. The first user asked to define the criteria couldn't give any because the story was not relevant to him and he recommended talking to another user. The second user gave the same answer and after following a chain of several users it was found that the story was actually not needed. Insisting on defining test cases first can thus also be a way of detecting unneeded requirements.

Wolfgang proceeded by giving a brief demo of the Zühlke Stuffplanner tool which allows story cards to be elaborated and managed through a web-based application.

Wolfgang's conclusion was that agile projects are actually strongly requirements centric; requirements are expressed as user stories, and these are central artefacts that are used throughout the project, for capturing stakeholders' needs, prioritizing those needs, estimating effort, tracking progress, and devising acceptance tests. He finished by discussing a series of issues that keep him awake at night: How to reconcile agile delivery with the common notion of fixed-scope, fixed-price contracts that many clients are more accustomed to or that are sometimes required in the case of public contracts? How to arrive at reliable cost estimates for an agile project? How to reconcile the self-organizing nature of agile teams with clients' preference for a project manager that provides a single point of contact? How to deal with users that are too busy to be involved in defining detailed user stories and acceptance criteria?

Paper Presentations

The workshop continued with presentations of 4 papers that had been accepted to the workshop after peer review. Each presentation was followed by lively discussions of the presented ideas.

The first paper "The Requirements Engineer as a Liason Officer in Agile Software Development" was presented by **Elke Hochmuller**. Elke described the considerable challenges faced by customers in agile projects and proposed that most of these challenges could be addressed by creating a role of liaison officer between customers and developers. The description of the requirements engineer as a liaison officer resonated with many of the practicing requirements engineers in the workshop. Elke also related this idea to a recent study by Angela Martin, Robert Biddle, and James Noble who identified different roles that members of customer organisations play in agile projects (geek interpreter, technical liaison, negotiator, customer coach, skill specialist).

The second paper "A Case Study on Benefits and Side-Effects of Agile Practices in Large-Scale Requirements Engineering" by **Elizabeth Bjarnason, Krzysztof Wnuk** and **Bjorn Regnell** was presented by the first author. Elizabeth presented the findings of interviews of 9 practitioners from a large-scale market-driven software development company. The purpose of the study was to find out their perception of the benefits and challenges of the agile practices they had recently adopted. While the study identified perceived benefits, particularly in addressing overscoping and communication problems, it also highlighted that agile practices had introduced new challenges such as finding a good balance between agility and discipline and ensuring sufficient competence within the teams.

Helen Sharp presented the third paper, "Communication Patterns of Agile Requirements Engineering" by **Nik Nailah Binti Abdullah, Shinichi Honiden, Helen Sharp, Bashar Nuseibeh** and **David Notkin**. She reported an ethnographically-informed study of requirements-related communications within a commercial agile team. The most obvious evidence of requirements in an agile team is provided by the user stories written on index cards displayed on the project wall. These cards however are mostly used to track progress and contain no or very little information about functionalities and goals. A story card is a promise of a conversation where the functionalities and goals will be clarified. The purpose of the study was to investigate in detail how communication and collaboration supports the understanding of requirements in this context. The study found evidence that through conversation and frequent re-articulation of requirements, the agile team members developed common concepts related to user stories, became aware of each others' goals, and that learning was taking place that allowed them to solve problems jointly. This suggests that a form of 'shared conceptualization' (a concept that Helen explained as being somewhat analogous to the psychological concept of 'flow' or 'being in the zone') underpins the team's requirements-related activities. This may have implications for agile practices in distributed teams: Is shared conceptualization reachable for distributed teams? Is it needed?

The fourth paper, "Analysing Requirements in a Case Study of Pairing" by **Yijun Yu** and **Helen Sharp** was presented by the first author. The idea put forward is that just as pair programming has proved to be efficient to support programming activities, "pair analysis" may be beneficial to requirements engineering activities. Yijun explained how this idea was tested for the OpenOME open source project – a project developing an i* goal modelling tool – where it was found that pairing practices for requirements engineering activities encouraged participants to assume a wider variety of roles in the project (researcher, developer, users) than without pairing.

Discussion

Presentations were followed by group discussions aimed at identifying open issues for agile requirements engineering. Several issues had come up during the presentations and ensuing discussions. These were extended and refined during a short brainstorming session with the whole group to include the following:

- *How to reconcile fixed-priced contracts with agile deliveries?*
- *How to obtain reliable cost-estimates for agile projects?*
- *How to reconcile developmers' desire for self-organisation and clients' desire for stricter management practices?*
- *People talk about having an agile mindset or having a requirements engineering mindest. What do these mean exactly?*
- *How to make agile requirements practices work for distributed teams?*
- *How to deal with limited stakeholders' availability?*
- *How to deal with quality requirements?*
- *Documentation: How much or how little requirements must be written down? What quality of requirements documentation is needed?*
- *How to find an adequate balance between agility and stability?*

We then split into two groups to prioritize these issues and discuss ways to make progress on them. The results were discussed again in plenary session. The two issues that were felt to be of most importance were those of documentation and distributed teams.

Requirements documentation in an agile project is limited to user stories and acceptance criteria, much of the detailed understanding of requirements is worked out during discussions. We noted that agile practices include requirements quality checks (for example the INVEST test presented in the keynote) that are similar to standard requirements quality criteria used for requirements reviews. The criteria used in agile projects put more emphasis on testability than is typically done in non-agile projects, but leave aside criteria of traceability, completeness, and absence of ambiguity. Leaving this aside, the group agreed that the key question is to identify how much documentation is needed and what qualities should be required of this documentation. Answering that question requires identifying why documentation would be needed for a project: Who is the documentation for? What purposes is it meant to serve? Documentation is only a means to other purposes. These include facilitating project understanding for new team members, earlier and cheaper detection of problems than by developing code, and acting as a contract between clients and developers. Some of these needs may be served by other means than traditional requirements documents (for example, pairing helps new team members become familiar with the application). The questions of how much documentation is needed must therefore be thought of in a wider context where purposes and other development practices are taken into consideration. The concept of requirements documentation itself could be given a wider acceptance than the traditional requirements document or database. It could for example include user stories, acceptance tests, email discussions, recordings of oral discussions in addition to more structured requirements documents. Issues of how to retrieve and make sense of information in such large volume of poorly structured data would then become important.

The second important issue concerned the handling of requirements in distributed agile projects. User stories and their associated annotations can be captured electronically and be made available through web-based systems. However, as Helen's presentation showed, much of the understanding that a team has about requirements come from frequent and spontaneous face-to-face discussions that may be difficult to reproduce in a distributed setting. Appropriate asynchronous means of communication may also be needed if teams are distributed over many time zones.

To conclude, the workshop provided an excellent opportunity to gain a better understanding of the relation between requirements engineering and agile development practice. There was a strong desire by those present to see a future edition of this workshop take place and an intention to hold it in an agile conference such as XP.

The workshop programme with slides for the presentation can be found here:

www.comp.lancs.ac.uk/~gacitur1/AREW11/index.htm

Emmanuel Letier
Joao Araujo
Ricardo Gacitua
Pete Sawyer

RE-marks

Templates for Thought

I recently set up a new website using the enormously popular WordPress platform. WordPress is the most widely used content managment system (CMS) on the internet, according to a recent W3Techs survey, used by 14.7% of the top million websites and with a CMS market share of 54.4%. The main WordPress site has a download counter which you can watch steadily tick by at the rate of about one download per second.

But WordPress is still just one of a large family of blogging tools and publishing platforms ranging from the quick and easy Tumblr and Posterous, through WordPress rivals Blogger, TypePad and LiveJournal, and up to business-scale CMS platforms such as Joomla and Drupal. And then there's the whole other corral of wiki platforms. The RESG site, for example, is based on MediaWiki.

As I was tinkering with the new WordPress set-up and reminding myself how to encourage those search spiders to crawl most efficiently around all the bits I wanted them to, I was increasingly struck by the underlying structural homogeneity of much of the web. Because they use such similar templates, a very large number of websites have more or less the same anatomy underneath their CSS skins: similarly-structured page hierarchies, content categories and layout. The next time you're browsing, count how many pages actually stand apart from the common mould in terms of their non-superficial design and structure. Even if a website isn't constructed from an off-the-shelf template, it's still likely to fit the general pattern because there is now a long-established way to design websites.

Of course, this structural homogeneity is a good thing, since it allows especially effective search engine optimisation (SEO). The finer points of SEO are something of a dark art, but beyond the overall structure of a site – which is imposed by default by the above platforms – there are general tips on how to phrase your written content and what styles of heading and caption work best (on the whole: be concise, plain, but descriptive). All of which makes a lot of sense. If you set-up a website, chances are you'd like as many people as possible to find it through searches for relevant related content.

But step back a moment. Is it always a good thing to be writing content and structuring a site primarily so it can be easily parsed by a program? What about all that freedom of expression and artful presentation? In the week I was playing around with WordPress, *New Scientist* coincidentally featured a piece called "Welcome to the 'algoworld'", which paints a picture of a world where misunderstood entities known as "algorithms" are running amok. But despite the alarmist vision, there were several sentences that chimed with my thoughts, such as this observation on recommender systems (such as Amazon uses): "The danger is that such an algorithm can create a monoculture. But this is not the way culture works – it is actually much spikier, much less predictable". And this, reiterating the above idea that search engines are the new patrons of our creativity: "It used to be that you wrote news for how people read – now it's written for how machines read ... It is shaping our expression and behaviour."

Requirements engineering is a discipline filled with frameworks and templates, each intended to elicit or capture human assumptions, intentions, and whims in efficient or exhaustive ways. There's no denying, generally speaking, that such frameworks are a boon to the requirements analyst or engineer. But are they always to the benefit of the activity? Do any of our readers have stories to support a view one way or the other?

I'd like to invite you share thoughts and observations: How important is it for a particular requirements framework or template – or notation, or modelling trope – to allow flexibility? Or, on the other hand, is it always more beneficial for tools of the requirements trade to impose artificial constraints or structure on the domain of human expression, though it may be rife with vacillation? Obviously, there's a balance but where does that balance lie?

If you have an opinion, observation, or anecdote please contact the editor – william.heaven@gmail.com – or leave a comment for this edition of *RQ* on the RESG website!

William Heaven

RE-flections

The Forgotten Keystone of Requirements Engineering

Soo Ling Lim

A newly launched expense system was rejected by its users. They were unaware of the project and the system was not meeting their needs. The system had to be redeveloped.

A railway timetabling website project was delayed for half a year. The website was completed on time, but a week before its scheduled release, the project manager found out to his horror that the new timetables needed approval. The approval took six months and should have been sought at the start of the project.

A team of developers spent the first year after their software was released handling change requests from users who had been ignored during requirements elicitation.

A medical software was banned from sales because it had not been approved by the health authority.

All these projects suffered from the same problem – their stakeholders were not identified correctly at the start. By stakeholders, I mean individuals or groups who can influence or be influenced by the outcome of

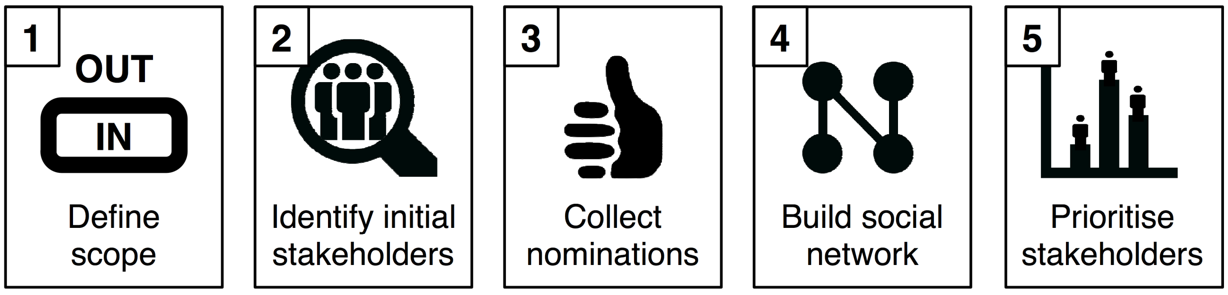
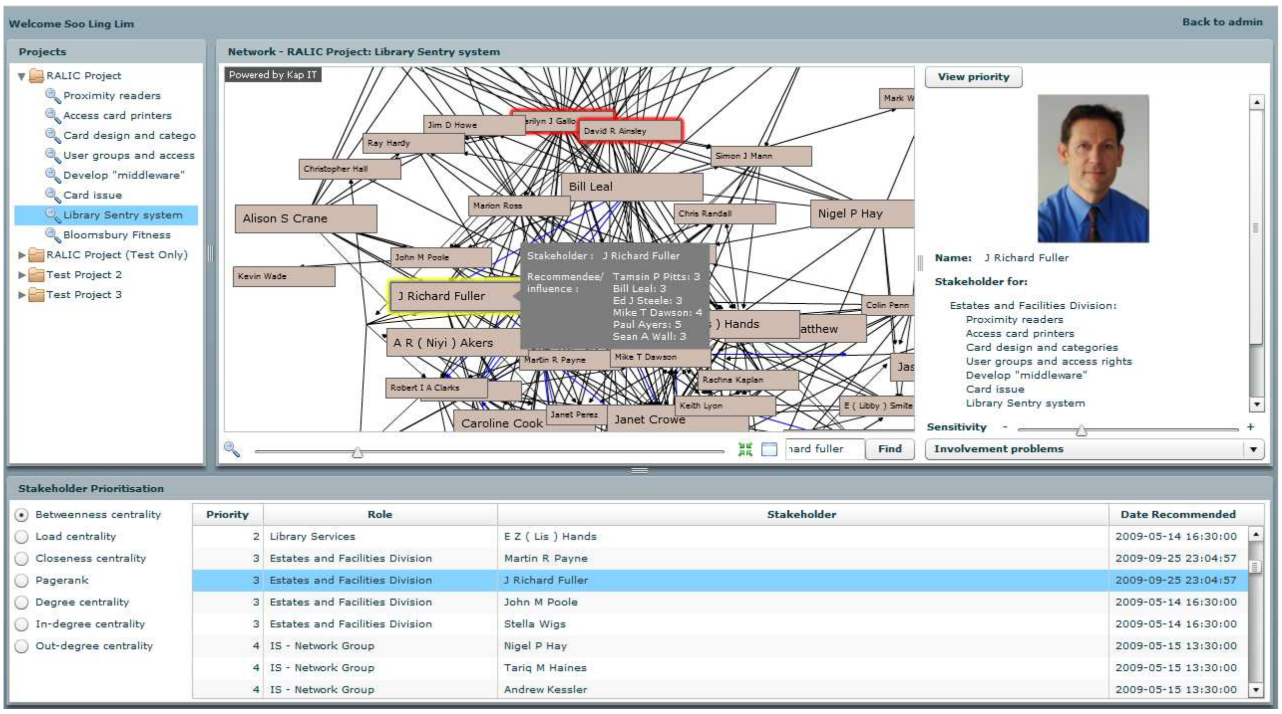


Fig. 1: Stakeholder Identification in Five Steps.



Stakeholder : J Richard Fuller

Recommendee/ influence : Tamsin P Pitts: 3
Bill Leal: 3
Ed J Steele: 3
Mike T Dawson: 4
Paul Ayers: 5
Sean A Wall: 3

Name: J Richard Fuller

Stakeholder for:

Estates and Facilities Division:
Proximity readers
Access card printers
Card design and categories
User groups and access rights
Develop "middleware"
Card issue
Library Sentry system

[✓ Betweenness centrality] Load centrality Closeness centrality Pagerank Degree centrality In-degree centrality Out-degree centrality

Priority	Role	Stakeholder	Date Recommended
2	Library Services	E Z (Lis) Hands	2009-05-14 16:30:00
3	Estates and Facilities Division	Martin R Payne	2009-09-25 23:04:57
3	Estates and Facilities Division	J Richard Fuller	2009-09-25 23:04:57
3	Estates and Facilities Division	John M Poole	2009-05-14 16:30:00
3	Estates and Facilities Division	Stella Wigs	2009-05-14 16:30:00
4	IS - Network Group	Nigel P Hay	2009-05-15 13:30:00
4	IS - Network Group	Tariq M Haines	2009-05-15 13:30:00
4	IS - Network Group	Andrew Kessler	2009-05-15 13:30:00

Fig. 2: Stakesource Network View

the software project.

Stakeholders play a central role in requirements elicitation. Requirements and project constraints come from stakeholders, so omitting stakeholders gives rise to missing requirements, which leads to building the wrong product. Some projects have only a handful of stakeholders; others can have hundreds or even hundreds of thousands of stakeholders. Some stakeholders have more influence, knowledge, or interest in the project than others. As such, the identification of suitable stakeholder representatives is crucial.

It is a mystery to me why stakeholders seem to be ignored or forgotten in RE. In a recent RESG poll by Camilo Fitzgerald [1], no one was interested in stakeholder identification! Yet in most of the projects I have seen, stakeholder identification is at worst non-existent and at best reactive – the analyst starts with an incomplete list of stakeholders and tackles new stakeholders as they appear throughout the project. In most cases, this reactive approach slows down the process and adds rework. But when an important stakeholder appears too late in the process, we have yet another horror story like the ones above.

This problem is well-known and has been described by many of the leaders of the field. Anthony Finkelstein listed stakeholder identification as one of the unsolved problems in requirements engineering in his 2004 AGM talk [2]. Ian Alexander and Suzanne Robertson held workshops called Stakeholders Without Tears [3]. I believe it is time for the RE community to tackle the problem of stakeholder identification head on – and that is why my research has focussed on this topic.

Social Networks For Stakeholder Analysis

My research uses ideas from the field of social network analysis. Stakeholders are socially related to one another. While each stakeholder only has a partial view of the project, together the group of stakeholders may hold a more complete picture than an individual expert. This new method involves all stakeholders in the stakeholder identification process. The analyst asks stakeholders to nominate other stakeholders, builds a social network of stakeholders, and prioritises the stakeholders using social network algorithms [4].

The method has five steps and is illustrated in Fig.1.

1. Define the scope of your project. Scope describes the boundary of the project, so that you know which stakeholders should be involved.
2. Based on this scope, identify an initial list of stakeholders.
3. Ask each stakeholder to nominate other stakeholders. Then, ask each newly identified stakeholder to nominate other stakeholders. This technique is also known as the snowballing technique. The identification of new stakeholders may result in the need to modify the scope. Feel free to do so and remember to adjust the stakeholder list accordingly.
4. Build a social network of stakeholders using the information collected in the previous step. Each node in the network represents a stakeholder.

Each link represents a nomination such that Node A links to Node B if stakeholder A nominates stakeholder B.

5. Apply social network algorithms to the stakeholder network. These algorithms rank stakeholders based on their position on the network (similar to how Google PageRank ranks websites). This step produces a prioritised list of stakeholders for your project.

The method was first tested on a project with a complex stakeholder base. The results were very exciting – I found that the method identified a comprehensive list of stakeholders. Not only that, but prioritisation using social network algorithms was more accurate compared to prioritisation from individual experts.

I was pleased at how well the method was received. Software companies were inviting me to talk about the method and share my experiences. Developers wanted their managers to use the method, and I was getting requests from project managers for a tool support. An IT director wanted to make the method a standard in his organisation. It was all very exciting. But manually collecting hundreds of nominations and processing them using the social network algorithms was something I could only do to test the method. For the method to become mainstream, most of the steps in the method can and should be automated.

My solution was StakeSource [5].

Stakesource

StakeSource is a Web2.0 tool that uses social networks and crowdsourcing for stakeholder identification [6]. It "crowdsources" the stakeholders for nominations, using their stake in the project to motivate their response. By using StakeSource, you only need to provide project details and initial stakeholders. StakeSource contacts the initial and newly identified stakeholders for nominations, builds the stakeholder network, applies the social network algorithms, and returns a prioritised list of stake-holders. You can access up-to-date information about the stakeholder network and prioritised list of stakeholders at any point during the process (see Fig. 2).

There was a lot interest in StakeSource. Less than a year of the prototype going live, more than ten projects had successfully used StakeSource – not bad for a system developed at the end of a PhD!

StakeSource continues to generate interest. We are meeting executives from various telecommunications, business analytics, and software companies. Most recently I have been invited to present StakeSource in the Ready-Set-Transfer game show at RE'11 organised by Jane Cleland-Huang and Daniela Damian. The game show will follow the format of Dragon's Den, with a panel of industry practitioners as the dragons.

Stakeholders are forgotten no longer in RE. While my work may just be the first steps towards a solution in stakeholder identification, I hope it will stimulate fresh interest in this important area.

[1] Requirements Quarterly 55 (August 2010)

[2] Anthony Finkelstien, "Unsolved Problems in Requirements Engineering", keynote at RESG AGM 2004.

[3] Ian Alexander and Suzanne Robertson (2004). Understanding project sociology by modeling stakeholders. *IEEE Software* 21(1), pp. 23-27

[4] Soo Ling Lim, Daniele Quercia, Anthony Finkelstein (2010). StakeNet: using social networks to analyse the stakeholders of large-scale software projects. *ICSE* (1) 2010, pp. 295-304

[5] www.stakesource.co.uk

[6] Soo Ling Lim, Daniele Quercia, and Anthony Finkelstein, StakeSource: harnessing the power of crowd-sourcing and social networks in stakeholder analysis. *ICSE* (2) 2010, pp. 239-242.

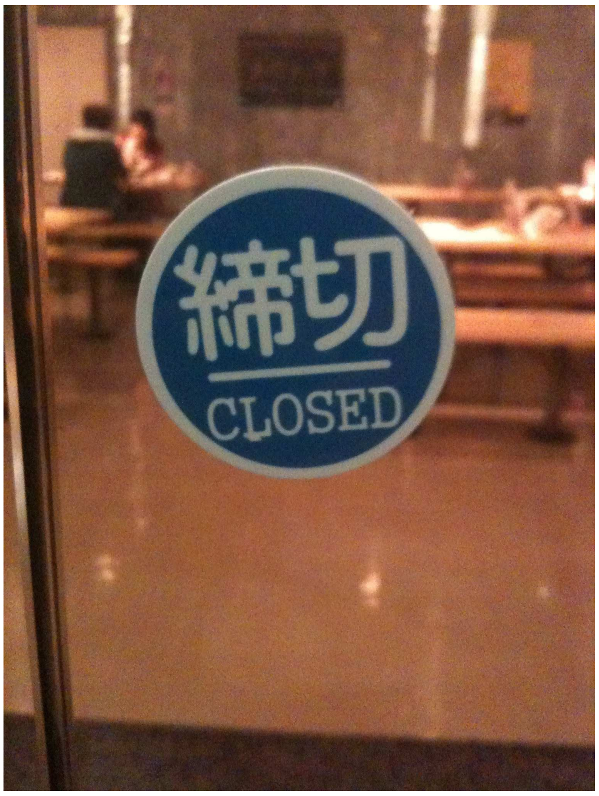
Soo Ling Lim is a Research Associate in the Department of Computer Science, University College London. She has a PhD in computer science and engineering from the University of New South Wales in Australia, and a bachelor of software engineering with first class honours from the Australian National University. Before her PhD, she worked as an ERP analyst programmer and a SAP consultant at the Computer Sciences Corporation and as a software engineer at CIC Secure in Australia. She specialises in stakeholder analysis, requirements elicitation, prioritisation, and change management, and offers her services as a consultant in these areas.

Soo Ling can be contacted at s.lim@cs.ucl.ac.uk.

Upfront Analysis: The Payoff

Camilo Fitzgerald

The question of how much upfront RE analysis to perform is one of considerable complexity that has always plagued software projects – proposed answers have come from both ends of the spectrum. Research in requirements engineering typically suggests that more needs to be done [1], and that the cost ratio of fixing defects at the requirements level to later stages is between 5 and 10 for smaller projects and between 10 and 100 for larger ones [2]. In practice, however,



Sign A - Restaurant Closed For Business?

many a manager says, "*I know that we should work out the requirements in detail, but we don't have time. We have to get started on the programming because we have a short deadline to deliver the code!*" [3]

The upfront analysis question is prevalent in many fields. When do we stop exploring a problem and begin taking action to solve it?

During my stay in Japan in the past six months I have seen many examples of signs designed to be understood by foreigners that illustrate the problem. Some of these have answered this question well, clearly communicating the desired message, some have done poorly, while others have simply left me baffled.

I encountered Sign A on the entrance to one of my favourite Tokyo restaurants. At first I was disappointed, thinking that I would have to satisfy my hunger somewhere else. I grew suspicious, however, when observing that the sign was permanently fixed to the door and on enquiring discovered that the establishment was in fact open. I later succeeded in translating the Japanese characters on the sign, revealing that its desired purpose was to state 'that the door should be kept closed'. The restaurant manager had not realised the more obvious interpretation of the sign, which I suspect may have lost the custom of many a foreigner.

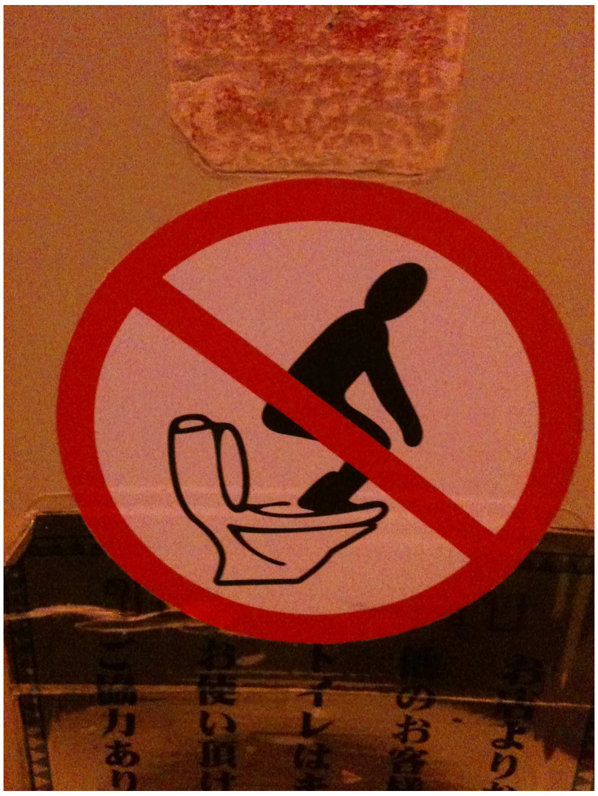
This sign is an all too common example of when a lack of upfront analysis can lead to unobserved negative consequences. Cutting corners on requirements analysis may result in a software product that performs its functions correctly, but in doing so has a strong negative impact on other business goals. Exploring negative impact with techniques like stakeholder analysis [4] can often result in an increase in product value that would otherwise be missed.

Signs B and C come from the walls of washrooms and tell contrasting stories. The first I found in a restaurant and was purchased from a bric-a-brac shop at the cost of a sticker. The shop owner hoped it would prevent clients accustomed to traditional Japanese toilets using their 'western-style' one in the same fashion.

In contrast, Sign C can be found in many metro stations and appears to have had a great deal spent on its sleek minimalist design and the mass manufacture of its metal cuttings. I first imagined it to signal a baby-changing station, but alas the sign appeared in many units where none was to be found. Its true meaning remains a riddle to me.

A comparison of these two signs exemplifies some of the factors to take into consideration when analysing the upfront analysis trade-off.

In the case of Sign B the costs of design and implementation, defects, and later fixes are low – buying and positioning a sticker, cleaning the toilet, and replacing the sticker if negative consequences



Sign B – Washroom Information

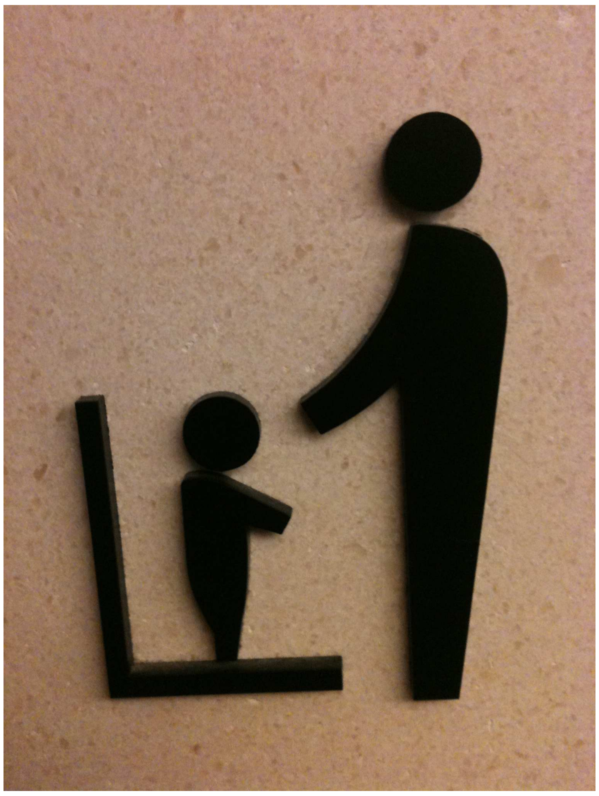
result. We can assume, therefore, that the restaurant manager might be right in putting little consideration into how effective the sign would be.

With Sign C, meanwhile, these costs are high – design, distribution and installation of the metallic cut-outs across the metro network, many customers failing to understand the desired message, and replacing all such signs. A rough estimate of these values might well have led to a strong case for further upfront analysis, such as extended assessment of the proposed sign's clarity on test subjects.

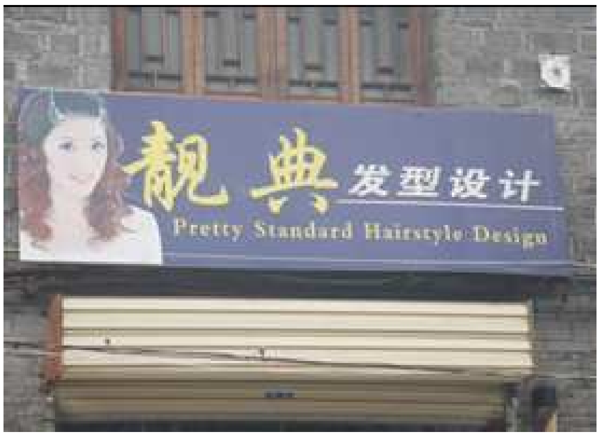
The cost of upfront analysis and the probability of its success also need to be taken into consideration. The key factors can be defined as:

- C_{imp}: cost of design and implementation.
- C_{def}: cost of product defects that could result from a lack of upfront analysis.
- C_{fix}: cost of fixing defects at a later date.
- P: probability that further upfront analysis will reduce later defects.
- C_{act}: cost of further upfront analysis.

The first four points relate to the benefits of upfront analysis, and the net value of a unit of upfront analysis can be estimated to be its benefit minus its cost. An oversimplified equation for calculating the net value of a unit of further upfront analysis is as follows:



Sign C – Washroom Information



Sign D - Upmarket Hair Salon

$$P \times (C_{def} + C_{fix} + C_{imp}) - C_{act}$$

Such an equation can provide an answer to the upfront analysis question. P will invariably decrease as more upfront analysis takes place, and upfront analysis should continue until the equation gives us a negative value.

In a complex software project it is a difficult task to construct the upfront analysis equation and estimate values for its variables that accurately model the realities of specific domains. Costs, for example, can have multiple measurement criteria such as man-hours, budget, damage to a product's reputation and loss of stakeholder satisfaction. Further, other factors may constrain the equation such as deadlines and the availability of manpower. The complexity of these models makes their construction and maintenance for software projects a non-trivial side-project in its own right.

If we were to have a pool of such equations that could be called on for specific domains this task would be made much easier. In a recent paper I provided such a cost model conducting extra upfront requirements analysis in online feature request management systems [5]. A set of ready-made models for other domains may well aid us in our efforts to answer the age-old upfront analysis question.

I leave you with Sign D from an upmarket hair salon in central China that tells a slightly different story. It goes to show that even when the individual components, or in this case words, of a solution are carefully constructed the solution as a whole can behave very differently. Admittedly though, foreigners in the area were few and far between.

[1] Daniela Damian and James Chisan, "An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity,

Quality, and Risk Management", IEEE Transactions on Software Engineering 2006

[2] S. McConnell, Code Complete: A Practical Handbook of Software Construction, volume 2, Microsoft Press, 2004

[3] Daniel Berry et al., "To do or not to do: If the requirements engineering payoff is so good, why aren't more companies doing it?", International Conference on Requirements Engineering, 2005

[4] Ian Alexander, "A Taxonomy of Stakeholders: Human Roles in System Development", International Journal of Technology and Human Interaction, 2005

[5] Camilo Fitzgerald et. al, "Early Failure Prediction in Feature Request Management Systems", International Conference on Requirements Engineering, 2011

Camilo Fitzgerald

RE-verberations

Nerdish Techy Stuff

Secondary-level technology education in the UK was recently subjected to both the high-profile criticism of Google CEO Eric Schmidt and the damning indictment of *The Observer*'s John Naughton in the space of a week.

Delivering the MacTaggart Lecture to a selection of the television industry's great and good at the Edinburgh International Television Festival, Schmidt candidly called us out on our current lacklustre standing in technology:

"If I may be so impolite, your track record isn't great. The UK is home of so many media-related inventions. You invented photography. You invented TV. You invented computers in both concept and practice.

"It's not widely known, but the world's first office computer was built in 1951 by Lyons' chain of tea shops. Yet today, none of the world's leading exponents in these fields are from the UK."

Schmidt pinned a large part of the blame on the fact that computer science was not a compulsory part of the curriculum. Also gone are the days of the BBC Micro, which arguably raised a generation of programmers and system engineers with its invitation (indeed, you might say *insistence*) to tinker. Instead, as Schmidt observed:

"Your IT curriculum focuses on teaching how to use software, but gives no insight into how it's made. That is just throwing away your great computing heritage."

John Naughton wasn't pulling punches either:

"What is happening is that the national curriculum's worthy aspirations to educate pupils about ICT are transmuted at the chalkface into teaching kids to use Microsoft software. Our children are mostly getting ICT training rather than ICT education.

"And if you can't see the difference, try this simple thought-experiment: replace 'ICT' with 'sex' and see which you'd prefer in that context: education or training?"

One of the suggestions Schmidt made was that art and science be brought back together, citing Lewis Carroll – both fairy-tale author and Oxford mathematician, of course – as the right kind of role-model for academic students.

But judging by some of the online reponses to Naughton's article a lot of people don't want to blur that line. One reader noted:

"Just as you can drive without being a mechanic, so you can use a computer without being a programmer. Schools need to teach lifeskills rather than esoteric academic subjects or nerdish techy stuff."

Other commenters were quick to point out that there are many excellent teachers and students in the UK today. As ever, the truth of the matter will be a lot less clear cut than the rhetoric of a newspaper piece allows.

Nonetheless, a strong basic understanding of technology is surely a requirement for all school-leavers today: this has now become a "lifeskill". As more and more aspects of life involve, or are managed by, software it's important to impart at least a small sense of what's going on behind the friend recommendations on Facebook, behind the ordering of search results by Google, or behind the global trading in financial markets. This is our world, after all.

It strikes me that a lot of this discipline-straddling – between the arts or humanities on one hand and technology and engineering on the other – is what we find in requirements engineering. Perhaps rather than raising a generation of programmers we might raise a generation of requirements engineers! Humanities students versed in technology and engineers versed in the arts. Would it not be a great bonus, with sophisticated software systems and computing infrastructure now underpinning most industries, for the complexities and subtleties in the construction of such systems to be more generally appreciated?

But if a generation of technologically literate business managers were to arrive how would the role of the requirements engineer change? Or, indeed, what if every stakeholder fancied herself a requirements engineer?

William Heaven

Sources

BBC News: www.bbc.co.uk/news/uk-14683133

Observer: www.guardian.co.uk/technology/2011/aug/28/ict-changes-needed-national-curriculum

RE-writings

What's fundamentally wrong? Improving our approach towards capturing value in requirements specification

Tom Gilb and Lindsey Brodie

We are all aware that many of our IT projects fail and disappoint: the poor state of requirements practice is frequently stated as a contributing factor. This article proposes a fundamental cause is that we think like programmers, not engineers and managers. We fail to concentrate on value delivery, and instead focus on functions, on use-cases and on code delivery. Our requirements specification practices fail to adequately address capturing value-related information. Compound-ing this problem, senior management is not taking its responsibility to make things better: managers are not effectively communicating about value and demanding value delivery. This article outlines some practical suggestions aimed at tackling these problems and improving the quality of requirements specification.

Introduction

We know many of our IT projects fail and disappoint, and that the overall picture is not dramatically improving [1] [2]. We also know that the poor state of requirements practice is frequently stated as one of the contributing failure factors [3] [4]. However, maybe a more fundamental cause can be proposed? A cause, which to date has received little recognition, and that certainly fails to be addressed by many well known and widely taught methods. What is this fundamental cause? In a nutshell: that we think like programmers, and not as engineers and managers. In other words, we do not concentrate on value delivery, but instead focus on functions, on use cases and on code delivery. As a result, we pay too little attention to capturing value and value-related information in our requirements specifications. We fail to capture the information that allows us to adequately consider priorities, and engineer and manage stakeholder-valued solutions.

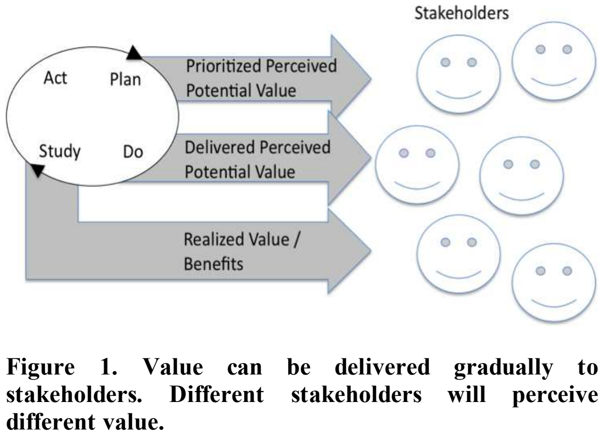

Stakeholders
Act
Plan
Study
Do
Prioritized Perceived Potential Value
Delivered Perceived Potential Value
Realized Value / Benefits

Figure 1. Value can be delivered gradually to stakeholders. Different stakeholders will perceive different value.

This article outlines some practical suggestions aimed at tackling these problems and improving the quality of requirements specification. It focuses on 'raising the bar' for communicating about value within our requirements. Of course, there is much still to be learnt about specifying value, but we can make a start – and achieve substantial improvement in IT project delivery – by applying what is already known to be good practice.

Note there is little that is new in what follows, and much of what is said can be simply regarded as commonsense. However, since IT projects continue not to grasp the significance of the approach advocated, and as there are people who have yet to encounter this way of thinking, it is worth repeating!

Definition of Value

The whole point of a project is achieving 'realized value' (also known as 'benefits'), for the stakeholders: it is not the defined functionality, and not the user stories that actually count. Value can be defined as 'the benefit we think we get from something' [5, page 435]. See Figure 1.

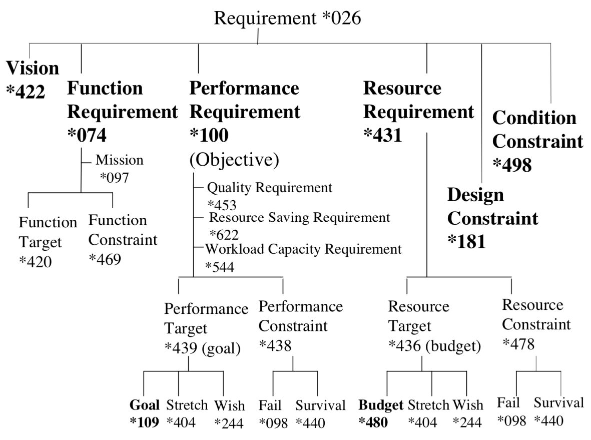

Requirement *026
Vision *422
Function Requirement *074
Performance Requirement *100 (Objective)
Resource Requirement *431
Condition Constraint *498
Design Constraint *181
Mission *097
Function Target *420
Function Constraint *469
Quality Requirement *453
Resource Saving Requirement *622
Workload Capacity Requirement *544
Performance Target *439 (goal)
Performance Constraint *438
Resource Target *436 (budget)
Resource Constraint *478
Goal *109
Stretch *404
Wish *244
Fail *098
Survival *440
Budget *480
Stretch *404
Wish *244
Fail *098
Survival *440

Figure 2. Example of Planguage requirements concepts

Ten Key Principles for Requirements Engineering

1. Understand the top level critical objectives
2. Think stakeholders: not just users and customers!
3. Focus on the required system quality, not just its functionality
4. Quantify quality requirements as a basis for software engineering
5. Don't mix ends and means
6. Capture explicit information about value
7. Ensure there is 'rich specification': requirement specifications need far more information than the requirement itself!
8. Carry out specification quality control (SQC)
9. Consider the total lifecycle and apply systems-thinking - not just a focus on software
10. Recognize that requirements change: use feedback and update requirements as necessary

Figure 3. Ten Key Principles for Requirements Engineering

Notice the subtle distinction between initially perceived value ('I think that would be useful'), and realized value: effective and factual value ('this was in practice more valuable than we thought it would be, because …'). Realized value has dependencies on the stakeholders actually utilizing a project's deliverables.

The issue with much of the conventional requirements thinking is that it is not closely enough coupled with 'value'. IT business analysts frequently fail to gather the information supporting a more precise understanding and/or the calculation of value. Moreover, the business people when stating their requirements frequently fail to justify them using value. The danger if requirements are not closely tied to value is that we lack the basic information allowing us to engineer and prioritize implementation to achieve value delivery, and we risk failure to deliver the required expected value, even if the 'requirements' are satisfied.

It is worth pointing out that 'value' is multi-dimensional. A given requirement can have financial value, environmental value, competitive advantage value, architectural value, as well as many other dimensions of value. Certainly value requires much more explicit definition than the priority groups used by MoSCoW ('Must Have', 'Should Have', 'Could Have', and 'Would like to Have/Won't Have This Time') [6] or by the Planning Game ('Essential', 'Less Essential' and 'Nice To Have') [7] for prioritizing requirements. Further, for an IT project, engineering 'value' also involves consideration of not just the requirements, but also the optional designs and the resources available: tradeoffs are needed. However, these are topics for future articles, this article focuses on the initial improvements needed in requirements specification to start to move towards value thinking.

Definition of Requirement

Do we all have a shared notion of what a 'requirement' is? This is another of our problems. Everybody has an opinion, and many of the opinions about the meaning of the concept 'requirement' are at variance: few of the popular definitions are correct or useful - especially when you consider the concept of 'value' alongside them. We have decided to define a requirement as a "stakeholder-valued end state". You possibly will not accept, or use this definition yet, but we have chosen it to emphasize the 'point' of IT systems engineering.

Example of Initial Weak Top-Level Critical Objectives

1. Central to the corporation's business strategy is to be the world's premier integrated <domain> service provider
2. Will provide a much more efficient user experience
3. Dramatically scale back the time frequently needed after the last data is acquired to time align, depth correct, splice, merge, recomputed and/or do whatever else is needed to generate the desired products
4. Make the system much easier to understand and use than has been the case with the previous system
5. A primary goal is to provide a much more productive system development environment then was previously the case
6. Will provide a richer set of functionality for supporting next generation logging tools and applications
7. Robustness is an essential system requirement
8. Major improvements in data quality over current practices

Figure 4. Example of Initial Weak Top-Level Critical Objectives

In previous work, we have identified, and defined a large number of requirement concepts [5, see Glossary, pages 321-438]. A sample of these concepts is given in Figure 2. You can use these concepts and the notion of a "stakeholder-valued end state" to re-examine your current requirements specifications. In the rest of this article, we provide more detailed discussion about some of the key points (the "key principles") you should consider.

The key principles are summarized in Figure 3. Let's now examine these principles in more detail.

Note, unless otherwise specified, further details on all aspects of Planguage (a planning language developed by one of the authors, Tom Gilb) can be found in [5].

Principle 1. Understand the top-level critical objectives

The 'worst requirement sin of all' is found in almost *all* the IT projects we look at, and this applies internationally. Time and again, the high-level requirements – also known as the top-level critical objectives (the ones that fund the project), are vaguely stated, and ignored by the project team. Such requirements frequently look like the example given in Figure 4 (which has been slightly edited to retain anonymity). These requirements are for a real project that ran for eight years and cost over 100 million US dollars. The project failed to deliver any of them. However, the main problem is that these are not top-level critical objectives: they fail to explain in sufficient detail what the business is trying to achieve: there are no real pointers to indicate the business aims and priorities. There are additional problems as well that will be discussed further later (such as lack of quantification, mixing optional designs into the requirements, and insufficient background description).

Management at the CEO, CTO and CIO level did not take the trouble to clarify these critical objectives. In fact, the CIO told me that the CEO actively rejected the idea of clarification! So management lost control of the project at the very beginning. Further, none of the technical 'experts' reacted to the situation. They happily spent $100 million on all the many suggested architecture solutions that were mixed in with the objectives.

It actually took less than an hour to rewrite one of these objectives, "Robustness", so that it was clear, measurable, and quantified (see later). So in one day's work the project could have clarified the objectives, and perhaps avoided some of the eight years of wasted time and effort.

Principle 2. Think stakeholders: not just users and customers!

Too many requirements specifications limit their scope to being too narrowly focused on user or customer needs. The broader area of *stakeholder* needs and values should be considered, where a 'stakeholder' is anyone or anything that has an interest in the system [5, page 420]. It is not just the users and customers that must be considered: IT development, IT maintenance, senior management, operational management, regulators, government, as well as other stakeholders can matter. The different stakeholders will have different viewpoints on the requirements and their associated value. Further, the stakeholders will be "experts" in different areas of the requirements. These different viewpoints will potentially lead to differences in opinion over the

Description of requirement/work task	**Past**	**Current Status**
Usability.Productivity: Time for the system to generate a survey	7200 sec	15 sec
Usability.Productivity: Time to set up a typical market research report	65 min	20 min
Usability.Productivity: Time to grant a set of end-users access to a report set and distribute report login information	80 min	5 min
Usability.Intuitiveness: The time in minutes it takes a medium-experienced programmer to define a complete and correct data transfer definition with Confirmit Web Services without any user documentation or any other aid	15 min	5 min
Performance.Runtime.Concurrency: Maximum number of simultaneous respondents executing a survey with a click rate of 20 sec and a response time < 500ms given a defined [Survey Complexity] and a defined [Server Configuration, Typical]	250 users	6000

Table 1. Extract from Confirmit Case Study [8]

implementation priorities.

Principle 3. Focus on the required system quality, not just its functionality

Far too much attention is paid to what the system must do (function) and far too little attention is given to how well it should do it (qualities). Many requirements specifications consist of detailed explanation of the functionality with only brief description of the required system quality. This is in spite of the fact that quality improvements tend to be the major drivers for new projects.

In contrast, here's an example, the Confirmit case study [8], where the focus of the project was not on functionality, but on driving up the system quality. By focusing on the "Usability" and "Performance" quality requirements the project achieved a great deal! See Table 1.

By system quality we mean all the "-ilities" and other qualities that a system can express. Some system developers limit system quality to referring to bug levels in code. However, a broader definition should be used. System qualities include availability, usability, portability, and any other quality that a stakeholder is interested in, like intuitiveness or robustness. See Figure 5, which shows a set of quality requirements. It also shows the notion that resources are "input" or used by a function, which in turn "outputs" or expresses system qualities. Sometimes the system qualities are mis-termed "non-functional requirements (NFRs)", but as can be seen in this figure, the system qualities are completely linked to the system functionality. In fact, different parts of the system functionality are likely to require different system qualities.

Principle 4. Quantify quality requirements as a basis for software engineering

Frequently we fail to practice "software engineering" in the sense of real engineering as described by engineering professors, like Koen [9]. All too often quality requirements specifications consist merely of words. No numbers, just nice sounding words; good enough to fool managers into spending millions for nothing (for example, "a much more efficient user experience").

We seem to almost totally avoid the practice of

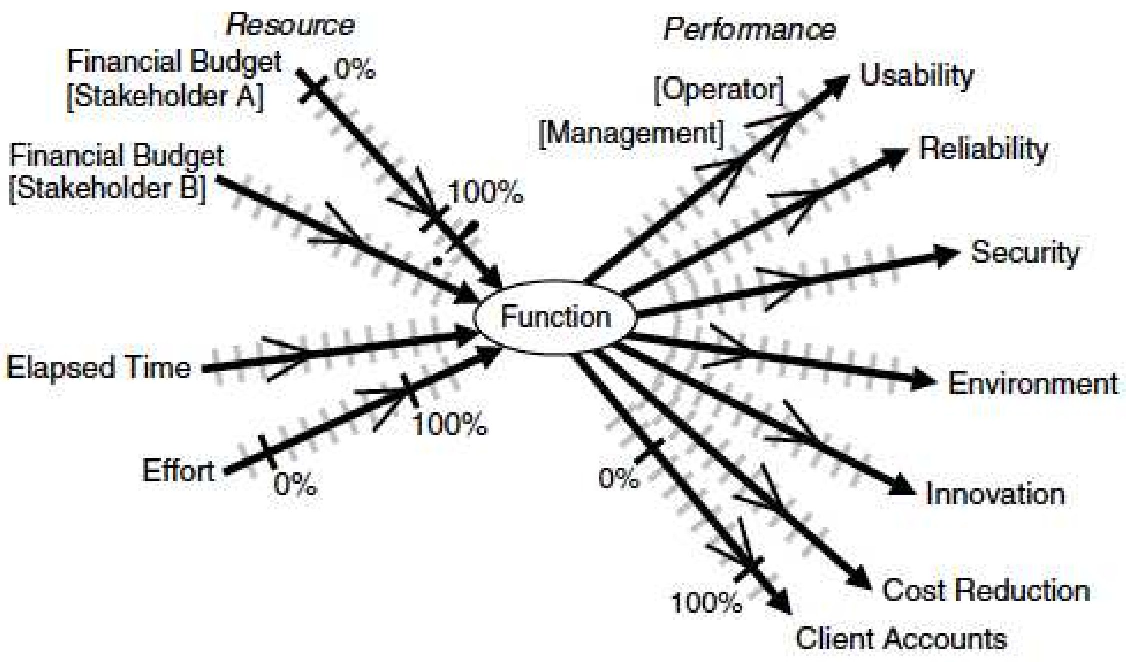


Figure 5. A way of visualizing qualities in relation to function and cost. Qualities and costs are scalar variables, so we can define scales of measure in order to discuss them numerically. The arrows on the scale arrows represent interesting points, such as the requirement levels. The requirement is not 'security' as such, but a defined, and testable degree of security [5, page 163]

Usability.Intuitiveness:

Type: Marketing Product Quality Requirement.

Ambition: Any potential user, any age, can immediately discover and correctly use all functions of the product, without training, help from friends, or external documentation.

Scale: % chance that defined [User] can successfully complete defined [Tasks] <immediately> with no external help.

Meter: Consumer reports tests all tasks for all defined user types, and gives public report.

Goal [Market = USA, User = Seniors, Product = New Version, Task = Photo Tasks Set, When = 2012]: 80% ±10% <- Draft Marketing Plan.

Figure 6. A simple example of quantifying a quality requirement, 'Intuitiveness'.

quantifying qualities. Yet we need quantification in order to make the quality requirements clearly understood, and also to lay the basis for measuring and tracking our progress in improvement towards meeting them. Further, it is the quantification that is the key to a better understanding of cost and value – different levels of quality have different associated cost and value.

The key idea for quantification is to define, or reuse a definition, of a scale of measure. For example, for a quality "Intuitiveness", a sub-component of "Usability":

To give some explanation of the key quantification features in Figure 6:

1. Ambition is a high-level summary of the requirement. One that is easy to agree to, and understand roughly.
2. Scale is the formal definition of our chosen scale of measure. The parameters [User] and [Task] allow us to generalize here, while becoming more specific in detail below (see later). They also encourage and permit the reuse of the Scale, as a sort of 'pattern'.
3. Meter provides a defined measuring process. There can be more than one for different occasions.
4. Goal is one of many possible requirement levels (see earlier detail in Figure 2 for some others: Stretch, Wish, Fail and Survival). We are defining a stakeholder-valued future state (for example: 80% ± 10%).

One *stakeholder* is 'USA Seniors'. The *future* is 2012. The requirement level type, Goal, is defined as a very high priority, budgeted promise of delivery. It is of higher priority than a Stretch or Wish level. Note other priorities may conflict and prevent this particular requirement from being delivered in practice.

If you know the *conventional* state of requirements methods, then you will now, from this example alone, begin to appreciate the difference proposed by such quantification - especially for *quality* requirements. IT projects already quantify time, cost,, response time, burn rate, and bug density – but there is much *more to achieve system engineering*!

Robustness:

Type: Complex Product Quality Requirement.

Includes: {Software Downtime, Restore Speed, Testability, Fault Prevention Capability, Fault Isolation Capability, Fault Analysis Capability, Hardware Debugging Capability}.

Figure 7. Definition of a complex quality requirement, Robustness

Testability:

Type: Software Quality Requirement.

Version: Oct 20, 2006.

Status: Draft.

Stakeholder: {Operator, Tester}.

Ambition: Rapid duration automatic testing of <critical complex tests> with extreme operator setup and initiation.

Scale: The duration of a defined [Volume] of testing or a defined [Type of Testing] by a defined [Skill Level] of system operator under defined [Operating Conditions].

Goal [All Customer Use, Volume = 1,000,000 data items, Type of Testing = WireXXXX vs. DXX, Skill Level = First Time Novice, Operating Conditions = Field]: < 10 minutes.

Design: Tool simulators, reverse cracking tool, generation of simulated telemetry frames entirely in software, application specific sophistication for drilling – recorded mode simulation by playing back the dump file, application test harness console <- 6.2.1 HFS.

Figure 8. Quantitative definition of Testability, an attribute of Robustness

Here is another example of quantification (see Figure 7). It is the initial stage of the rewrite of Robustness from the Figure 4 example. First we determined that Robustness is complex and composed of many different attributes, such as Testability.

Note this example shows the notion of there being different levels of requirements. Principle 1 also has relevance here as it is concerned with top-level objectives (requirements). The different levels that can be identified include: corporate requirements, the top-level critical few project or product requirements, system requirements and software requirements. We need to clearly document the level and the interactions amongst these requirements.

An additional notion is that of 'sets of requirements'. Any given stakeholder is likely to have a set of requirements rather than just an isolated single requirement. In fact, achieving value could depend on meeting an entire set of requirements.

Principle 5. Don't mix ends and means

"Perfection of means and confusion of ends seem to characterize our age." *Albert Einstein. 1879-1955*

The problem of confusing ends and means is clearly an old one, and deeply rooted. We specify a solution, design and/or architecture, instead of what we really value – our real requirement. There are explanatory reasons for this – for example solutions are more concrete, and what we want (qualities) are more abstract for us (because we have not yet learned to make them measurable).

The problems occur when we do confuse them: if we do specify the means, and not our true ends. As the saying goes: "Be careful what you ask for, you might

Why do you require a 'password'? For Security!

What kind of security do you want? Against stolen information.

What level of strength of security against stolen information are you willing to pay for? At least a 99% chance that hackers cannot break in within 1 hour of trying! Whatever that level costs up to €1 million.

So that is your real requirement? Yep.

Can we make that the official requirement, and leave the security design to both our security experts, and leave it to proof by measurement to decide what is really the right design? Of course!

The aim being that whatever technology we choose, it gets you the 99%?

Sure, thanks for helping me articulate that!

Figure 9. Example of the requirement, not the design feature, being the real requirement

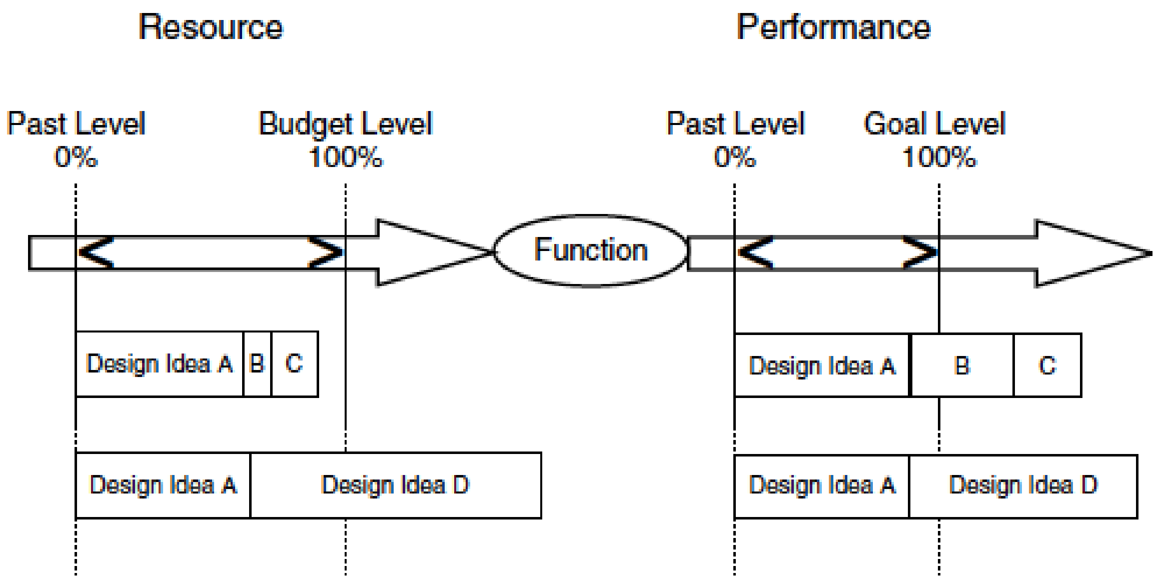


Figure 10. A graphical way of understanding performance attributes (which include all qualities) in relation to function, design and resources. Design ideas cost some resources, and design ideas deliver performance (including system qualities) for given functions. Source [5, page 192].

just get it" (unknown source). The problems include:

- You might not get what you *really* want.
- The solution you have specified might *cost too much* or have bad *side effects*, even if you do get what you want.
- There may be much *better solutions* you don't know about yet.

So how to we find the 'right requirement', the 'real requirement' [10] that is being 'masked' by the solution? *Assume* that there probably is a better formulation, which is a more accurate expression of our real values and needs. Search for it by asking 'Why?' Why do I want X, it is because I really want Y, and assume I will get it through X. But, then why do I want Y? Because I really want Z and assume that is the best way to get X. Continue the process until it seems reasonable to stop. This is a slight variation on the '5 Whys' technique [11], which is normally used to identify root causes of problems (rather than high-level requirements).

Assume that our stakeholders will *usually* state their values in terms of some perceived means to get what they really value. Help them to identify (The 5 Whys?) and to acknowledge what they really want, and make that the 'official' requirement. Don't insult them by telling them that they don't know what they want. But explain that you will help them more-certainly get what they more deeply want, with better and cheaper solutions, perhaps new technology, if they will go through the '5 Whys?' process with you. See Figure 9.

Note that this separation of designs from the requirements does not mean that you ignore the solutions/designs/architecture when software engineering. It is just that you must separate your requirements - including any mandatory means - from any optional means. The key thing is to understand what is optional so that you consider alternative solutions. See Figure 10, which shows two alternative solutions: Design A with Designs B and C, or Design A with Design D. Assuming that say, Design B was mandatory, could distort your project planning.

Principle 6. Capture explicit information about value

How can we articulate and document notions of value in a requirement specification? See the example for Intuitiveness, a component quality of Usability, given in Figure 11, which expands on Figure 6.

Usability.Intuitiveness:

Type: Marketing Product Requirement.

Stakeholders: {Marketing Director, Support Manager, Training Center}.

Impacts: {Product Sales, Support Costs, Training Effort, Documentation Design}.

Supports: Corporate Quality Policy 2.3.

Ambition: Any potential user, any age, can immediately discover and correctly use all functions of the product, without training, help from friends, or external documentation.

Scale: % chance that a defined [User] can successfully complete the defined [Tasks] <immediately>, with no external help.

Meter: Consumer Reports tests all tasks for all defined user types, and gives public report.

----Analysis --------------------------------------

Trend [Market = Asia, User = {Teenager, Early Adopters}, Product = Main Competitor, Projection = 2013]: 95%±3% <- Market Analysis.

Past [Market = USA, User = Seniors, Product = Old Version, Task = Photo Tasks Set, When = 2010]: 70% ±10% <- Our Labs Measures.

Record [Market = Finland, User = {Android Mobile Phone, Teenagers}, Task = Phone+SMS Task Set, Record Set = January 2010]: 98% ±1% <- Secret Report.

----Our Product Plans -------------------------------

Goal [Market = USA, User = Seniors, Product = New Version, Task = Photo Tasks Set, When = 2012]: 80% ±10% <- Draft Marketing Plan.

Value [Market =USA, User = Seniors, Product = New Version, Task = Photo Tasks Set, Time Period = 2012]: 2M USD.

Tolerable [Market = Asia, User = {Teenager, Early Adopters}, Product = Our New Version, Deadline = 2013]: 97%±3% <- Marketing Director Speech.

Fail [Market = Finland, User = {Android Mobile Phone, Teenagers}, Task = Phone+SMS Task Set, Product Release 9.0]: Less Than 95%.

Value [Market = Finland, User = {Android Mobile Phone, Teenagers}, Task = Phone+SMS Task Set, Time Period = 2013]: 30K USD.

Figure 11. A fictitious Planguage example, designed to display ways of making the value of a requirement clear

For brevity, a detailed explanation is not given here. Hopefully, the Planguage specification is reasonably understandable without detailed explanation. For example, the Goal statement (80%) specifies which market ("USA") and users ("Seniors") it is intended for, which set of tasks are valued (the "Photo Tasks Set"), and when it would be valuable to get it delivered ("2012"). This 'qualifier' information in all the statements, helps document where, who, what, and when the quality level applies. The additional Value parameter specifies the perceived value of achieving 100% of the requirement. Of course, more could be said about value and its specification, this is merely a 'wake-up call' that explicit value needs to be captured within requirements. It is better than the more common specifications of the Usability requirement, that we often see, such as: "The product will be more user-friendly, using Windows".

So who is going to make these value statements in requirements specifications? I don't expect developers to care much about value statements. Their job is to deliver the requirement levels that someone else has determined are valued. Deciding what sets of requirements are valuable is a Product Owner (Scrum) or Marketing Management function. Certainly, the IT staff should only determine the value related to IT stakeholder requirements!

Principle 7. Ensure there is 'rich specification': requirement specifications need far more information than the requirement itself!

Far too much emphasis is often placed on the requirement itself; and far too little concurrent information is gathered about its background, for example: who wants this requirement and when? The requirement itself might be less than 10% of a complete requirement specification that includes the background information. It should be a corporate standard to specify this related background information, and to ensure it is intimately and immediately tied into the requirement itself.

Such background information is *useful* related information, but is not *central* (*core)* to the implementation, and nor is it commentary. The central information includes: Scale, Meter, Goal, Definition and Constraint.

Background specification includes: benchmarks {Past, Record, Trend}, Owner, Version, Stakeholders, Gist (brief description), Ambition, Impacts, and Supports.

The rationale for background information is as follows:

- To help judge the value of the requirement
- To help prioritize the requirement
- To help understand the risks associated with the requirement
- To help present the requirement in more or less detail for various audiences and different purposes
- To give us help when updating a requirement
- To synchronize the relationships between different but related levels of the requirements
- To assist in quality control of the requirements
- To improve the clarity of the requirement.

Commentary is any detail that probably will not have any economic, quality or effort consequences if it is incorrect, for example, notes and comments.

See Figure 12 for an example, which illustrates the help given by background information regarding risks.

Reliability:
Type: Performance Quality.
Owner: Quality Director. **Author:** John Engineer.
Stakeholders: {Users, Shops, Repair Centers}.
Scale: Mean Time Between Failure.
Goal [Users]: 20,000 hours <- Customer Survey, 2004.
Rationale: Anything less would be uncompetitive.
Assumption: Our main competitor does not improve more than 10%.
Issues: New competitors might appear.
Risks: The technology costs to reach this level might be excessive.
Design Suggestion: Triple redundant software and database system.
Goal [Shops]: 30,000 hours <- Quality Director.
Rationale: Customer contract specification.
Assumption: This is technically possible today.
Issues: The necessary technology might cause undesired schedule delays.
Risks: The customer might merge with a competitor chain and leave us to foot the costs for the component parts that they might no longer require.
Design Suggestion: Simplification and reuse of known components.

Figure 12. A requirement specification can be embellished with many background specifications that will help us to understand risks associated with one or more elements of the requirement specification [12].

TEMPLATE FOR FUNCTION SPECIFICATION <with hints>

Tag: <Tag name for the function>.
Type: <{Function Specification, Function (Target) Requirement, Function Constraint}>.

============= **Basic Information** ===============================
Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: <Name the role/email/person responsible for changes and updates to this specification>.
Stakeholders: <Name any stakeholders with an interest in this specification>.
Gist: <Give a 5 to 20 word summary of the nature of this function>.
Description: <Give a detailed, unambiguous description of the function, or a tag reference to someplace where it is detailed. Remember to include definitions of any local terms>.

============= **Relationships** ==
Supra-functions: <List tag of function/mission, which this function is a part of. A hierarchy of tags, such as A.B.C, is even more illuminating. Note: an alternative way of expressing supra-function is to use Is Part Of>.
Sub-functions: <List the tags of any immediate sub-functions (that is, the next level down), of this function. Note: alternative ways of expressing sub-functions are Includes and Consists Of>.
Is Impacted By: <List the tags of any design ideas or Evo steps delivering, or capable of delivering, this function. The actual function is NOT modified by the design idea, but its presence in the system is, or can be, altered in some way. This is an Impact Estimation table relationship>.
Linked To: <List names or tags of any other system specifications, which this one is related to intimately, in addition to the above specified hierarchical function relations and IE-related links. Note: an alternative way is to express such a relationship is to use Supports or Is Supported By, as appropriate>.

============= **Measurement** ==
Test: <Refer to tags of any test plan or/and test cases, which deal with this function>.

============= **Priority and Risk Management** ==========================
Rationale: < Justify the existence of this function. Why is this function necessary? >.
Value: <Name [Stakeholder, time, place, event>]: <Quantify, or express in words, the value claimed as a result of delivering the requirement>.
Assumptions: <Specify, or refer to tags of any assumptions in connection with this function, which could cause problems if they were not true, or later became invalid>.
Dependencies: <Using text or tags, name anything, which is dependent on this function in any significant way, or which this function itself, is dependent on in any significant way>.
Risks: <List or refer to tags of anything, which could cause malfunction, delay, or negative impacts on plans, requirements and expected results>.
Priority: <Name, using tags, any system elements, which this function can clearly be done *after* or must clearly be done *before*. Give any relevant reasons>.
Issues: <State any known issues>.

============= **Specific Budgets** ====================================
Financial Budget: <Refer to the allocated money for planning and implementation (which includes test) of this function>.

Figure 13. A template for function specification [5, page 106]

Background information must not be scattered around in different documents and meeting notes. It needs to be directly integrated into a sole master reusable requirement specification object. Otherwise it will not be available when it is needed: it will not be updated, or shown to be inconsistent with emerging improvements in the requirement specification.

Principle 8. Carry out specification quality control (SQC)

There is far too little quality control of requirements against relevant standards. All requirements specifications ought to pass their quality control checks before they are released for use by the next processes. Initial quality control of requirements specification, where there has been no previous use of specification quality control (SQC) (also known as Inspection), using three simple quality-checking rules ('unambiguous to readers', 'testable' and 'no optional designs present'), typically identifies 80 to 200+ words per 300 words of requirement text as ambiguous or unclear to intended readers! [13]

Principle 9. Consider the total lifecycle and apply systems-thinking - not just a focus on software

If we don't consider the total lifecycle of the system, we risk failing to think about all the things that are necessary prerequisites to actually delivering *full value* to real *stakeholders* on time. For example, if we want better maintainability then it has to be designed into the system. If we are really engineering costs, then we need to think about the total operational costs over time. This is much more than just considering the programming aspects.

You must take into account the nature of the system: an exploratory web application doesn't need to same level of software engineering as a real-time banking system!

Principle 10. Recognise that requirements change: use feedback and update requirements as necessary

Ideally requirements must be developed based on on-going feedback from stakeholders, as to their real value. System development methods, such as the agile methods, enable this to occur. Stakeholders can give feedback about their perception of value, based on the *realities* of actually using the system. The requirements must be *evolved* based on this realistic experience. The whole process is a 'Plan Do Study Act' Shewhart cyclical learning process involving many complex factors, including factors from outside the system, such as politics, law, international differences, economics, and technology change.

Attempts to fix the requirements in advance of feedback, are typically wasted energy (unless the requirements are completely known upfront, which might be the case in a straightforward system rewrite with no system changes). Committing to fixed requirements specifications in contracts is not realistic.

See Figure 13 for a requirement template for function specification [5, page 106], which hints at the richness possible for background information.

Who Or What Will Change Things?

Everybody talks about requirements, but few people seem to be making progress to enhance the quality of their requirements specifications and improve support for software engineering. Yes, there are internationally competitive businesses, like HP and Intel that have long since improved their practices because of their competitive nature and necessity [8, 14]. But they are very different from the majority of organizations building software. The vast majority of IT systems development teams we encounter are not highly motivated to learn or practice first class requirements (or anything else!). Neither the managers nor the systems developers seem strongly motivated to improve. The reason is that they get by with, and even get well paid for, failed projects.

The universities certainly do not train IT/computer science students well in requirements, and the business schools also certainly do not train managers about such matters [15]. The fashion now seems to be to learn oversimplified methods, and/or methods prescribed by some certification or standardization body. Perhaps insurance companies and lawmakers might demand better industry practices, but I fear that even *that* would be corrupted in practice if history is any guide (for example, think of CMMI and the various organization certified as being at Level 5).

Summary

Current requirements specification practice is often woefully inadequate for today's critical and complex systems. Yet we do know a considerable amount (Not all!) about good practice. The main question is whether your 'requirements' actually capture the true breadth of information that is needed to make a start on engineering value for your stakeholders.

Here are some specific questions for you to ask about your current IT project's requirements specification:

- *Do you have a list of top-level critical objectives?*
- *Do you consider multiple stakeholder viewpoints?*
- *Do you know the expected stakeholder value to be delivered?*
- *Have you quantified your top five quality attributes? Are they are testable? What are the current levels for these quality attributes?*
- *Are there any optional designs in your requirements?*
- *Can you state the source of each of your requirements?*
- *What is the quality level of your requirements documentation? That is, the number of major defects remaining per page?*
- *When are you planning to deliver stakeholder value? To which stakeholders?*

If you can't answer these questions with the 'right' answers, then you have work to do! And you might also better understand why your IT project is drifting from delivering its requirements. The good news is that the approach outlined in this article should allow you to focus rapidly on what really matters to your stakeholders: value delivery.

References

1. Thomas Carper, Report Card to the Senate Hearing "Off-Line and Off-Budget: The Dismal State of Federal Information Technology Planing", July 31, 2008. See uscpt.net/CPT_InTheNews.aspx [Last Accessed: August 2010]

2. The Standish Group, "Chaos Summary 2009", 2009. See www.standishgroup.com/newsroom/chaos_2009.php [Last Accessed: August 2010]

3. John McManus and Trevor Wood-Harper, "A Study in Project Failure", June 2008. See www.bcs.org/server.php?show=ConWebDoc.19584 [Last Accessed: August 2010].

4. David Yardley, *Successful IT Project Delivery*, Addison-Wesley, 2002.

5. Tom Gilb, "Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering using Planguage", Elsevier Butterworth-Heinemann, 2005.

6. Jennifer Stapleton (Editor), *DSDM: Business Focused Development* (2nd Edition), Addison Wesley, 2003.

7. Mike Cohn, *User Stories Applied: For Agile Software Development*, Addison Wesley, 2004.

8. Trond Johansen and Tom Gilb, From Waterfall to Evolutionary Development (Evo): How we created faster, more user-friendly, more productive software products for a multi-national market, Proceedings of INCOSE, 2005. See www.gilb.com/tiki-download_file.php?fileId=32

9. Dr. Billy Vaughn Koen, "Discussion of the Method: Conducting the Engineer's Approach to Problem Solving", Oxford University Press, 2003.

10. Tom Gilb, Real Requirements, see www.gilb.com/tiki-download_file.php?fileId=28

11. Taiichi Ohno, "Toyota production system: beyond large-scale production", Productivity Press, 1988.

12. Tom Gilb, "Rich Requirement Specs: The use of Planguage to clarify requirements", see www.gilb.com/tiki-download_file.php?fileId=44

13. Tom Gilb, Agile Specification Quality Control, Testing Experience, March 2009. Download from www.testingexperience.com/testingexperience01_08.pdf [Last Accessed: August 2010].

14. Top Level Objectives: A slide collection of case studies. See www.gilb.com/tiki-download_file.php?fileId=180

15. Kenneth Hopper and William Hopper, "The Puritan Gift", I. B. Taurus and Co. Ltd., 2007.

Tom Gilb
Independent Consultant
Tom@Gilb.com

Lindsey Brodie
Middlesex University
L.Brodie@mdx.ac.uk

RE-sources

Books & Papers

RQ archive at the RESG website:
www.resg.org.uk

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

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

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

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

IFIP Working Group 2.9 (Software RE):
eeat.cis.gsu.edu/ifip2.9/

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

Volere template:
www.volere.co.uk

Lists & Online Communities

RESG LinkedIn Group

www.linkedin.com/groups/Requirements-Engineering-Specialist-Group-RESG-2662234

RESG Mailing List
www.resg.org.uk/mailing_list.html

RE-online
discuss.it.uts.edu.au/mailman/listinfo/re-online

ReQuirements Networking Group
www.requirementsnetwork.com

RE Yahoo Group
groups.yahoo.com/group/Requirements-Engineering/

RE-actor

The committee of the RESG

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

Chair (outgoing):
Ian Alexander,
Scenario plus,
iany@scenarioplus.org.uk

Chair (incoming):
Emmanuel Letier
University College London,
e.letier@cs.ucl.ac.uk

Treasurer:
Steve Armstrong,
The Open University,
s.armstrong@open.ac.uk

Secretary:
James Lockerbie
City University London,
ac769 @ soi.city.ac.uk

Membership Secretary:
Yijun Yu
The Open University
y.yu@open.ac.uk

Publicity Officer:
Camilo Fitzgerald
University College London,
c.fitzgerald@cs.ucl.ac.ukk

***RQ* Editor**:
William Heaven
Freelance Writer / Consultant,
william.heaven@gmail.com

Newsletter Reporter:
Ljerka Beus-Dukić
University of Westminster
l.beus-dukic@wmin.ac.uk

Student Officer:
Ben Jennings
University College London
b.jennings@cs.ucl.ac.uk

Post Graduate Officer:
Dalal Alrajeh
Imperial College London
dalal.alrajeh@imperial.ac.uk

Academic Member:
Peter Sawyer,
Lancaster University,
sawyer@comp.lancs.ac.uk

Academic Member:
Prof. Bashar Nuseibeh
The Open University
b.nuseibeh@open.ac.uk

Industrial Member:
Alistair Mavin,
Aero Engine Controls,
alistair.mavin@rolls-royce.com

Industrial Member:
Vesna Music,
Delphi Diesel Systems,
vesna.music@delphi.com

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

Regional Officer:
Shehan Gunewardene,
CAP Gemini/Aspire,
shehan.gunawardena@hmrcaspire.com

Your Name Here!
Please contact any of the committee if you'd like to join us.

Contributing to RQ

To contribute to *RQ* please send contributions to William Heaven (william.heaven@gmail.com). Submissions must be in electronic form, preferably as plain text.

The deadline for RQ 58 (December 2011) is 1 December 2011

Joining the RESG

Visit www.resg.org.uk/index.php/Join for membership details, or email Yijun Yu (Y.Yu@open.ac.uk).