BCS

Requirenautics Quarterly

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

http://www.resg.org.uk

©2001, BCS RESG Issue 24 (July 2001)

RE-Locations

RE-Soundings 1
Editorial ***1***
Chairman's message ***1***

RE-Treats 2
Systems Available to All ***2***

RE-Calls 2
International Council on Systems Engineerig 12th Annual Symposium (INCOSE 2002)**, July 18 – August 1, 2002, Las Vegas, Nevada, USA.* ***2

RE-Readings 2
Using Win-Win for Requirements Negotiation**, Imperial College, 5 April 2001.* ***2

RE-Papers 3
A Vision for Requirements Engineering ***3***
In Praise of Scope Creep ***6***
On Abstraction in Scenarios ***9***

RE-Publications 10
Book Reviews ***10***

RE-Sources 13
Mailing lists ***13***

RE-Actors 13
The committee of RESG ***13***

RE-Soundings

Editorial

In many ways RE practice changes only slowly; that's how it should be for an activity so critical to the development process. But change it does as new techniques are developed and validated and, crucially, as existing good practice is disseminated.

One of the means for achieving dissemination is to publish books and there has been a small flood of books on RE and related themes in recent years. The last few months has seen a further three added to publishers' lists and this RQ contains timely reviews of them. We're doubly lucky because to complement the reviews two of the authors, Richard Veryard and Ralph Young, have written articles outlining their 'vision' for RE.

The third book is by our very own Michael Jackson (Patron of the RESG) and regular readers, or indeed anyone involved in RE, will be aware of Michael's work. If not, see Michael's article in RQ23.

Barry Boehm is someone else whose work has had a huge influence on software and systems engineering. We were lucky to have Barry and his colleague Paul Grünbacher visit Imperial College to deliver an RESG seminar on *Win-Win* a few months ago. As always, Ian Alexander was there to record the event for the benefit of those who couldn't make it.

So if you haven't yet taken your vacation and the schoolboy wizard genre isn't your thing, maybe this issue of RQ will save you having to scour the airport bookshop for something to read by the pool.

Pete Sawyer
Computing Department, Lancaster University.

Chairman's message

I'm writing this introduction in the sweltering heat of summer in London, so I'll keep it short! There are a few weeks to go for RE'01 in Toronto, but the symposium will probably be over by the time you receive this copy of RQ. The RESG is a co-sponsor of RE'01, and some members of the RESG committee have been involved in the organisation and programme committees of the symposium. It has been in the planning for about two years now, and the last few months have been particularly hectic as we put the final programme together. Check it out at www.re01.org.

Closer to home, the RESG organised a successful mini-tutorial on requirements negotiation, run by Barry Boehm and Paul Grunbacher. The event, combining tutorial presentation and hands-on exercises using the EasyWinWin automated tool, proved to be a very interesting and enjoyable afternoon.

Due to unforeseen circumstances to do with a shortage of resources, we had to postpone our July meeting on Requirements for Product Families. Since the AGM was also planned to precede this meeting, we decided to postpone the AGM as well. It will now planned for November - the exact date to be announced on the emailing list. The consequence of this is that you have even more time to consider volunteering to join the RESG Executive Committee! Do drop me a line if you're interested in joining the RESG team.

Best wishes for a relaxing and productive summer.

Bashar Nuseibeh
The Open University

RE-Treats

Next event organised by the group

Systems Available to All

Date: 19th September
Location: The Key Centre, University of Hertfordshire, Hatfield Campus.
Contact: David Shearer, Department of Computer Science, University of Hertfordshire.
(d.w.shearer@herts.ac.uk)

The Disability Rights Commission has amongst its responsibilities to promote the equalisation of opportunities for disabled persons. If software developers ignore the needs and requirements of disabled users, could their employers be liable to prosecution? This meeting is a joint meeting between the BCS Disability and Requirements Engineering Specialist Groups to consider how developers can take into account the needs and requirements of all potential users.

RE-Calls

Recent Calls for Papers and Participation

International Council on Systems Engineerig 12th Annual Symposium (INCOSE 2002), July 18 – August 1, 2002, Las Vegas, Nevada, USA.

http://www.incose.org/symp2002/

The Symposium theme "Engineering 21st Century Systems: Problem Solving through Structured Thinking" calls on delegates to discuss and debate the application of defined systems engineering processes to understand and solve the challenges that lie ahead in the definition and the design of new products and systems, the deployment of new technology, and the effective use and support of legacy systems that are still with us in this new century. The Symposium will provide a forum to address all aspects of 21 st Century problems and opportunities, the systems engineering principles, processes, methodologies and tools that can address them, and the resulting systems, products, and services.

Papers are invited that can make a contribution to the theme of the Symposium, within the framework of the traditional technical tracks and areas of special interest. All types of papers will be considered, including case studies, developmental work and technical analysis. Papers will be judged on a range of parameters including clarity of expression, effective communication of ideas, and technical content. Papers must be submitted in English, the official language of the INCOSE 2002 Symposium.

RE-Readings

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

Using Win-Win for Requirements Negotiation, Imperial College, 5 April 2001.

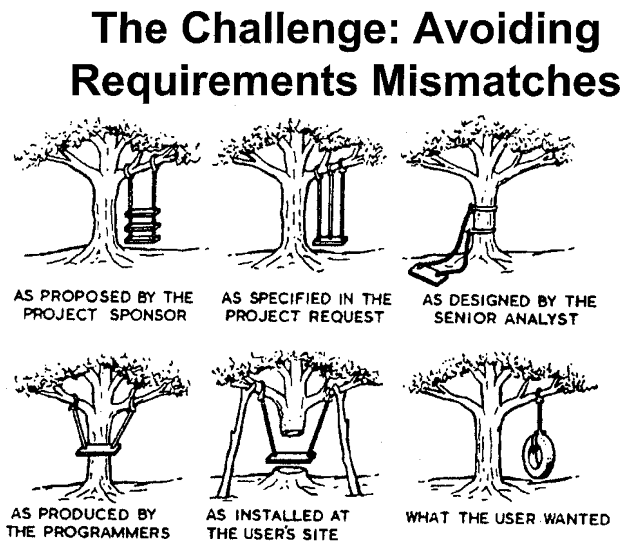
We had the great pleasure of welcoming one of the truly great names in software engineering to London in April: **Barry Boehm**, famous for COCOMO, the spiral life-cycle model, and much else. His latest baby is WinWin, and we were lucky enough to hear about it from the man himself. He appeared as a relaxed and gently humorous speaker, wearing his experience and erudition lightly – one of his first slides was the cartoon of the child's swing development project.

Barry Boehm said that his group now had a tool good enough for its own group planning exercises. There was a well worn path via the Standish Group's statistics showing that over half the trouble on development projects was requirements-related.

There was no complete and well-defined set of requirements waiting to be discovered, like some buried chest of gold sovereigns from Stevenson's Treasure Island. Requirements emerged from a process of co-operative learning, involving exploration, prioritization, negotiation, evaluation, and finally documenting the results. The clear implication was that skimping on any of these steps – trying to double-guess the users and writing their requirements for them – was bound to lead to trouble of one kind or another.

Requirements had to be negotiated to achieve a mutually satisfactory agreement, a "win-win". Why? Because the usual win-lose (I win, you lose) is unstable, and turns readily to lose-lose: the attempt to gain more than is fair results in losses all round. For example if a business oppresses its customers, they will eventually desert it.

Mutual satisfaction cannot be mutually optimal – the world is not like that. But win-win is the equilibrium state: it is defined as the state where there are no outstanding issues, and all the win conditions are covered by agreements.


The Challenge: Avoiding Requirements Mismatches
AS PROPOSED BY THE PROJECT SPONSOR
AS SPECIFIED IN THE PROJECT REQUEST
AS DESIGNED BY THE SENIOR ANALYST
AS PRODUCED BY THE PROGRAMMERS
AS INSTALLED AT THE USER'S SITE
WHAT THE USER WANTED

So, if you have no issues, and then add one condition, you have to get everyone to agree it: you re-enter the win-win process. This memorably "builds a team as well as a set of requirements".

The aim therefore was to make everyone a winner – this was the whole basis of systems engineering, he said.

Paul Grünbacher then took over and introduced the tool, known as EasyWinWin. 90 minutes of interaction was the limit for groupwork before taking a break. Enthusiasm rose for the first ten minutes, dipped, rose again as the facilitator said 'just five minutes more' and then faded out.

"It takes about an hour to sort out convergence on 100 conditions", claimed Grünbacher boldly. "People are really good at sorting [lists into order]". The procedure was therefore to brainstorm ideas, to sort these into predefined buckets, and to ask people to vote on conflicted items by judging a) the ease of implementing each one and b) its importance.

We had a go at brainstorming a list of the 100 best films of all time – it was fun and took just a few minutes – and then were frustrated by network problems: the server had been told to log users off if their browsers were idle for more than 2 minutes, and when everyone tried to reconnect at the same moment, we couldn't. A requirement on stress-testing the network performance had possibly been missed. However, the approach looks sensible and practical, and it addresses a problem that few other approaches have really faced – how to attain consensus, quickly. Other approaches either ignore it altogether or, like VORD, lay down a programme of conflict identification and resolution that is too lengthy to appeal to many development project managers. WinWin has now been tried out on several real projects including the $100M STARS project at DARPA, so it deserves a wider audience.

RE-Papers

A Vision for Requirements Engineering

Ralph Young

Pete Sawyer suggested that I write an article that sets out my vision for requirements engineering. My vision for requirements engineering is: 1) satisfied customers, 2) systems and software engineers who are fulfilled by their work, 3) trained and experienced requirements analysts, 4) projects completed on schedule and within budget, and 5) less than 15% rework and less than 10% wasted resources on development projects. My intent in this paper is to suggest some opportunities for requirements engineers to improve project success. These improvement opportunities go beyond the practices I discuss in *Effective Requirements Practices*, which also contains ideas, suggestions, and recommendations that will help practitioners address this vision.

In the Epilogue of my book, I assert that "The time is here for practitioners in the systems and software development profession to decide that we stand for excellence." Striving for excellence should be our standard for daily living. Our chance to have an impact is short--why not work each of our days to move us in that direction?

Figure 1 provides some opportunities for requirements engineers/analysts to improve project success rates. Think of these as enablers for the excellence we seek. More fundamentally, experience confirms that they are necessary conditions to achieve excellence in requirements engineering.

Requirements engineering is difficult. It is not just a simple matter of writing down what the customer wants. A fundamental problem in our business is that requirements are inherently dynamic-they will change over time as our understanding of the problem we are trying to solve changes. The importance of good requirements, and the underlying dynamic nature of the process, means that we must be as accurate as possible, and yet be flexible. Flexible does not mean "weak," but rather than we have a process for sculpting requirements that can change as we clarify the real requirements of our customers.

This is not a job for the most junior member of the team or the least talented member of the group. It requires merging of excellent technical skills with excellent domain knowledge and excellent people and communications skills.

Some of the challenges that we face in improving requirements engineering are:

-Project managers who are focused on daily activities and are unable to adequately address underlying human needs and long term issues. A recent industry study concluded that 75% of significant root causes of project failures are attributable to project planning and tracking.[1] Some PMs provide only lip service to quality, teamwork, and continuous improvement, and are unwilling to invest in training and practice to create a "quality improvement culture. [2]"

-Developers who refuse to use improved techniques even when these techniques have demonstrated they are known to yield better results, who have not learned requirements engineering techniques, and who have not been provided with expectations concerning how they are to work and what they are to do in different situations.

-Customers who believe they understand their needs (but let their ego force them to stake out an unmovable position), are impatient for short-term results, require use of risky techniques, and have limited experience working in a partnering[3] or team mode that requires common commitment to project success.

All of these challenges hinder excellence as our standard for daily living, teamwork as our approach to excellence, and continuous improvement as our habit.

Now we[4] have a set of challenges we can work on! And now we must decide that we are committed to pursuing them. Here are some examples of things that we can do in our daily work to live out our commitment.

1. Recognize that a productive work environment means supporting our people, achieving effective teamwork, and establishing a value of continuous improvement. Take the actions needed to create a productive work environment so that requirements-related work is effective.
2. (Development) projects must be managed (continuous oversight to ensure that the right things are being done properly and well). We must manage projects better and by doing so, reduce requirements-related defects.
3. Train and use specialists in requirements engineering.
4. Have and use a requirements process that addresses full life-cycle requirements-related activities. Invest 8-14% of total project effort in requirements-related activities throughout the project life cycle.
5. Spend more time discovering the *real* requirements.
6. Use effective requirements practices. Utilize a continuous improvement environment to get ever better.
7. Use an automated requirements tool to support the development effort.
8. Provide an effective process and mechanism to manage changes to requirements.
9. Take action when things aren't working. We know when things aren't working. Ensure that requirements-related efforts are effective.
10. Provide role models that consistently demonstrate effective work habits and disciplines, for example, creating and using a joint team to be responsible for the requirements.

Figure 1. Opportunities for Requirements Engineers To Improve Project Success Rates

How can we help project managers?

We can assist project managers (PMs) in benefiting from project experience. For example, most PMs in my experience would benefit from paying more attention to the human dimensions of projects. PMs could ask performers how things are going, and then reflect on and incorporate input whenever possible. The point is that there is a wealth of information and advice available from the staff of any project that can further strengthen and improve the practices being used on the project! People put forward their best efforts when they are fulfilled and feel empowered[5] in their

[1] PRC, 2001.

[2] A good reference is Karl E. Wiegers, *Creating a Software Engineering Culture*.

[3] See Frank Carr et al, *Partnering in Construction: A Practical Guide to Project Success* for a thorough treatment of the partnering process.

[4] By "we" in this article, I refer to people who are involved in performing requirements-related activities on systems and software development projects.

[5] By "empowered," I mean that employees feel responsible not only for doing a good job, but also for making the whole organization work better. Teams work together to improve their performance continually, achieving higher levels of productivity. Organizations are structured in such a way that people

work environment. PMs need to find out what the team members want to do, encourage ideas for improvement, and seek advice on how the PM can help to make their work environment more positive. They must create an atmosphere of trust and true interest in the opinions of team members and not be defensive about negative or unexpected feedback. Then, PMs must take actions to ensure the needed work is getting done *and* that people leave work most days feeling that they accomplished things.

Another thing we can do for project managers is to provide resources about industry experience concerning requirements engineering and make specific recommendations to them about approaches that they might consider for a specific project. For example, the data from the U.S. National Aeronautics and Space Administration (NASA) provided in *Effective Requirements Practices* (p. 62) give a clear and powerful message: Projects that expend the industry average of 2-3% of total project cost/effort on the (full life-cycle) requirements process experience an 80-200% cost overrun, while projects that invest 8-14% of total project cost/effort in the requirements process experience less than a 60% overrun. Obviously, our goal is to not have overruns at all; however, a smaller overrun is preferable to a larger one! Further, we need to clarify for our PMs that a portion of the one-third of project costs that are now wasted[6] can be redirected to pay for needed improvements in processes and practices. We need to show them how to track the return-on-investment from specific improvements, so that they have the data to make good decisions.

Our projects need an effective process and mechanism to manage changes in requirements. We know from our experience that this is where we lose control of many technical efforts. Let's take action to not let this happen again. For example, utilizing a joint customer/supplier team to maintain responsibility for the requirements throughout the development effort and having all change requests pass through a single channel will help. Another mechanism is controlling the sources of unofficial requirements. Documenting the rationale for each requirement will also reduce the number of requirements (by as much as 50%, according to industry expert Ivy Hooks).

In addition, we must teach project managers how to manage. If we define successful project management as providing an effective system, within budget, and on-time delivery, the *CHAOS Report*[7] documents that we don't manage projects successfully. Effective project management is possible--we need to learn how and to teach our PMs.[8]

How can we help our customers?

We need to educate our customers (and PMs) that the initial set of written requirements is seldom the set of *real* requirements. Customers need help from skilled requirements analysts to reveal the real requirements. The investment made to discover the real requirements would be more than repaid by avoiding downstream development work on an inadequate set of requirements. Let's not continue to re-learn this lesson on every future project!

Are we really harnessing the power of effective teamwork? We need to establish commitment to one another to allow any endeavor (not only our projects, but also our marriage, our children, and getting our children through college, for example) to be successful. We need to find creative ways to make teamwork happen, even in environments that don't support it. For example, an Integrated Product Team (IPT) approach in which the customer is a member of the IPT facilitates communication and team effort. We should advise our customers, PMs, and organizations of successful approaches that have achieved effective teamwork and nurture the commitment that is required for any project to be successful.

How can we help our developers?

One way in which we can assist our developers is to listen to them, and take action on what they say. Let them know that we "are there for them." By this I mean that it is management's job to provide a work environment that maximizes effectiveness. The bottom line is that our success depends upon our people. Whatever is produced and created is done by them. We must provide them an environment in which they are effective and fulfilled, where they can grow, where they hear from us regularly that they are appreciated and valued. We must care deeply, and show it.

Other ways in which we can assist our developers are to put effective processes and practices in place and expect them to be applied, to work at reducing rework (as mentioned previously, average rework on projects in our industry is 45%), to create an environment of continuous improvement, and to work toward ever-better quality of our work products. If we don't do this,

feel that they are able to achieve the results they want, that they can do what needs to be done, not just what is required of them. See Cynthia D. Scott and Dennis T. Jaffee, *Empowerment: A Practical Guide for Success.*

[6] As documented in *Effective Requirements Practices* (p. 79), 80% of all product defects are inserted in the requirements definition activities. Rework costs are estimated at 45% of total costs. By taking 80% of 45%, we learn that 36% (more than one-third) of total project costs (based on industry data) potentially can be avoided by driving requirements errors out of the work products.

[7] The Standish Group International, 1995.

[8] Two excellent books that provide effective project management techniques are Steve McConnell, *Software Project Survival Guide* and Fergus O'Connell, *How to Run Successful Projects II.*

we risk people becoming frustrated and leaving our organization and projects.

Still another important thing we must do is make it clear to our developers that we expect them to use the improved and proven policies, processes, methods, techniques, and tools that are the standards in our organization and on our projects. Watts Humphrey's "*Why Don't They Practice What We Preach*"[9] details some reasons developers do not use improved techniques even when they are provided with evidence that they achieve better results. We must make it clear that it's not acceptable for our people to fall back on their old ways of doing things. We need to provide role models that consistently demonstrate effective work habits and disciplines.[10] Einstein commented that the only rational way of educating is to be an example.

Whoever on our project is addressing requirements issues needs support. In addition to an effective automated requirements tool, requirements engineers need training in requirements engineering. What is a good requirement? Why must requirements analysts not make requirements decisions? Why must I not goldplate (add features and capabilities that are not required)? We must create a set of expectations for our project staff concerning how they are to work and what they are to do in different situations. This can best be accomplished by training that is tailored to a particular project's environment and needs, and presented by people who can really help us. The training and our advice to our staff must be done in a way that respects people--we all have egos, and if mine is hurt, it's difficult for me to put forward my best efforts.

In summary, there are many things we can do to create a pathway to address requirements-related problems. I'm not pretending that this is easy, nor that it can be accomplished quickly. Achieving our vision for requirements engineering requires, however, that we do things differently.

Are you ready?

[9] Available at http://www.sei.cmu.edu/publications/articles/practice-preach/practice-preach.html

[10] Steve McConnell's *After the Gold Rush: Creating a True Profession of Software Engineering* is full of ideas and suggestions to help with this situation. McConnell notes in his epilogue that common development problems won't be avoided without our support.

RE-Creations

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

Deadline for next issue: 1st October 2001.

In Praise of Scope Creep

Richard Veryard

You must know the story of Jack and the Beanstalk. Jack and mother poor, cow to market, Jack swaps cow for five beans, mother furious, beans thrown out of window, beanstalk grows, Jack climbs, giant wakes, fee-fi-fo-fum, goose lays, golden eggs, harp sings, beanstalk chopped down, giant dead, Jack rich.

But did you ever hear the story about Jack's uncle, the IT manager. He's still out there in the garden, looking for the other four beans.

An important aspect of project/programme management is scoping – the separation between what is included and what is excluded. Honest project managers rightly want to control costs and timescales of engineering activities, and they often do this by attacking a common devil known as Scope-Creep. Meanwhile, some dishonest project managers welcome Scope-Creep as an ally.

Scope-Creep has many other names. When he fiddles with the problem, he is called Requirements-Creep; when he fiddles with the solution he is called Feature-Creep; when he tries to engage extra stakeholders he is called Log-Roller; and when he tries to seduce stakeholders by promising extra benefits he is called Benefits-Scrounger.

But hold on, you might say. By condemning Scope-Creep as a devil, are we not being unfair to the poor old chap. Of course, there may always be some tediously narrow-minded project managers who are incapable of seeing him in a positive light, but he has a heart of pure gold. He has many close friends, who help him at all times, and I'm sure they will be happy to speak up for him.

Scope Creep in Project Management

In popular culture and belief, vampires can be kept at bay with garlic and the sign of the cross. In the mythology of project management, Scope-Creep can be kept at bay with magical charms and rituals such as change control procedures.

Honest project managers enact these charms and rituals with great seriousness. The project scope is guarded defensively, and any interaction with the outside world – whether incoming or outgoing – is regarded with some suspicion, lest it introduce additional responsibilities and obligations onto the project.

Dishonest project managers are often covertly in league with Scope-Creep, and enact the same charms and rituals cynically and corruptly. For them, a key tactic is to pass off legitimate requirements as manifestations of Scope-Creep, and they accumulate them as bargaining tokens in order to improve their resource–responsibility ratio. Nothing delights them more than an opportunity to grab extra budget while simultaneously stacking up excuses for failure. They make extravagant promises, and then rely on Scope-Creep to get them off the hook.

Government departments are often seriously ripped off by large contractors, who can use Scope-Creep as a way of escalating the size of the project and escaping from penalty clauses. Meanwhile, the contractors themselves see strict change control and contract adherence as a way of protecting themselves against Scope-Creep.

But hold on, you might say. By condemning Scope-Creep as a devil, are we not being unfair to the poor old chap. Of course, there may always be some tediously narrow-minded project managers who are incapable of seeing him in a positive light, but he has a heart of pure gold. He has many close friends, who help him at all times, and I'm sure they will be happy to speak up for him.

Scope-Creep has the soul of a true engineer. He is always trying to improve things, to perfect things. But unlike many engineers, he is also strongly committed to meeting the user's requirements. Whenever a project gets out of alignment with the requirements, Scope-Creep appears as if by magic, earnestly determined to put things right.

In a dynamic situation, aligning a project with the business requirements demands a degree of change. If the project manager tries to eliminate Scope Creep from a project, this amounts to a suppression of change, and a refusal to align with the business. In some cases, an inordinate amount of energy is devoted to resisting Scope-Creep, thereby undoing much of the bridge-building and rapport with users that was painstakingly established at the start of the project.

Although the project manager can prevent this consuming scheduled project time, she cannot always prevent it leaking commitment, as well as reducing the efficiency and quality of project-user interactions.

Indeed, our cynical project manager may use change control disciplines in order to increase misalignment and prevent the project doing anything useful. This is because a genuinely useful project usually generates a lot of criticism – firstly from the people who want it to succeed, and will crawl over every deliverable to make sure that it's all going to work properly; and secondly from the people who don't think it should have been done in the first place, and will be looking for reasons to get the project cancelled. If the project is obviously going to produce something of little value, whose relevance to the business has been significantly eroded by change, then nobody's going to bother much with the details, and the project team is left to get on with it.

Scope Creep in Process Improvement

As a consultant, I have frequently been faced with project managers who have strongly resisted my suggestions for improving their projects, because they regarded any change as a sign of Scope-Creep. Dear reader, you may imagine that I am eager to construct arguments why I was right and they were wrong. But I don't want to talk about myself, but about boundaries and interfaces.

The project manager sits on the boundary of a project, protecting the activity inside the boundary, making demands from people outside the boundary, managing communications across the boundary, and resisting all attempts to move the boundary. From the project manager's perspective, the boundary is an interface of a particular kind, which she perceives in a particular way.

As a consultant, I may want to support the project manager, but I don't do this by accepting without question the project manager's view of the world. And this includes the project manager's focus on scope. There are other important aspects of the project boundary besides scope, and it is often useful for the consultant to focus on these aspects instead, to provide a view that is complementary to the project manager's view.

As the project manager perceives it, what the consultant has proposed is usually to add something to the project, in order to make the whole thing smaller and/or more effective. To the sceptical project manager, this seems implausible if not paradoxical. Meanwhile, the consultant may not perceive the suggestion as adding anything at all. How can we account for this difference in perception?

What the consultant sometimes hears is: We're too busy to get things right first time. And the consultant then responds: So when are you going to have time to repeat it until it's right? But of course, that's not necessarily what the project manager thought she said. This is a fairly common pattern of misunderstanding between consultant and project manager. One way of viewing this misunderstanding is to hypothesize that the consultant simply doesn't see the boundary of the project in the same way as the project manager does. (This hypothesis, of course, takes a perspective outside both project manager and consultant. Whose perspective is this?)

After all, we consultants have an agenda too: we usually want to add value in some way to a project, preferably in a way that is visible to all concerned. And it isn't always only the project manager that we're trying to please or impress.

Scope Creep and Scope Trading in Programme Management

When we look at a large programme, consisting of many projects, the opportunities for Scope-Creep are greatly enhanced. The projects typically occupy neighbouring plots of land, like small gardens or allotments – and it's very tempting for the project manager to toss unwanted stuff over the fence into a neighbouring plot, or to steal attractive stuff.

Even honest project managers do this – not out of mischief but sometimes out of confusion, and sometimes in the belief that shifting material from one project to another benefits the whole programme. Each project team has a vision of its scope, and perhaps a vision for the whole programme from its own viewpoint – leading it to draw conclusions about the "proper" place for this or that material. Even when two neighbouring projects agree about the boundary between them, a system architect with a different perspective over the whole programme may wish to overrule them.

In risk management, it is the risks falling into the no-mans-land between the projects that often cause the greatest trouble. Here again, Scope Creep is at work, encouraging the risks to creep across the project boundaries like weeds encroaching your garden. This phenomenon might seem to give project managers some incentive to manage these risks collaboratively.

Programme management is therefore sometimes called upon to play a policing role – patrolling the boundaries to make sure the architect's vision is respected, and is not undermined or corrupted by scope-trading.

Scope and Identity

In order to understand where things belong in an organization, one needs a model of the organization. Such a model may be a public model in a formal notation, a private (and possibly unconscious) mental model, or something between these two extremes. Models are rarely complete or consistent, even formal ones. In practice, each participant may have his own slightly different model of the organization, and requirements engineering often fosters debates in which differences between the different models are exposed and clarified.

Requirements engineering is often dominated by a conceptual legacy. People are often influenced by the names of legacy systems - they interpret the name of a given system as a sign that this system ought to be the right place for a given requirement. Because this system is called the Billing System, it ought to handle everything that we regard as Billing. This conceptual legacy strongly affects the scoping of the Billing System and its boundaries with other systems. It may even affect the very integrity and survival of an entity called The Billing System - in other words, its identity.

Furthermore, scoping is usually based on some notion of clustering strongly related requirements together. But that depends what we perceive as strongly related - and these perceptions change over time. Many systems engineers find it easier to perceive new connections - new forms of relationship - than to forget old ones.

If our notion of the identity and scope of the Billing System gets broader and more complex over time, this can be regarded as a form of conceptual Scope Creep - perhaps even Vision Creep. Are our ideas becoming confused and bloated? Or are they expanding, growing, evolving, becoming enriched? That depends on your point of view.

Users' mental models of legacy systems and other artefacts are important in another way as well. The actual use of a system by actual users depends on their mental models of the system. The users often don't use all the features of the legacy software, and engineers may sometimes be pleased to discover that a new requirement can be satisfied by switching on an existing feature and retraining the users, with no need to restructure the legacy code. This could be seen as changing the socio- part of a sociotechnical legacy system, and leaving the technical part the same.

The engineers responsible for developing a system often try to anticipate undesired usages of "their" system by wayward users. Sometimes the system is heavily optimized towards a particular notion of its use, and alternative notions must be discouraged or banned, to avoid troublesome performance. (For example, complex online enquiries may be banned, because they would absorb too much network resources.)

But the users generally spend much longer using the system than the engineers spend designing it. And the users are generally more ingenious and determined than the engineers give them credit for. So they often find ways around the barriers, forcing the system to do what they want after all, albeit inefficiently.

Thus Scope Creep appears here in the form of Usage Drift - as new modes of using the artefact diffuse across a community of users. In general, all manifestations of Scope Creep - including Requirements Creep, Feature Creep and Usage Drift - can be understood as forms of diffusion. So how are we to position ourselves in relation to this diffusion - as passive recipients or as actors in some network?

Instruments of Scope

But does Scope Creep really exist?

It is no accident that SCOPE is used to designate various instruments supposed to enhance vision - telescope, microscope, periscope. Even with the best lenses, these instruments are imperfect, unreliable, with false or double images at the margins. And in requirements engineering also, scoping follows the logic of the visual field - it is based on drawing meaningful distinctions and regions across an **image** or **mirror** of the requirements (usually known as a model). Zooming in on a crisp line reveals it as a blurred region demanding further analysis - and this can be repeated ad infinitum, fractal style. Zooming out reveals separate regions and shapes and patterns that are not clearly visible in close-up. The tools and techniques of requirements engineering and project management are (imperfect) instruments for creating and viewing these images.

This leads to the conclusion that Scope Creep is an **imaginary** creature. Scope Creep can only be viewed through these instruments, may be a useful fiction, but has no independent real existence. We may project our problems onto this imaginary creature - but after all what would we be without such projections?

This article is an expanded extract from Chapter 4 of Richard Veryard. Component-Based Business: Plug and Play. Springer London, 2001

On Abstraction in Scenarios

Ian Alexander

How Abstract should Scenarios be? Or how vivid with 'real' detail? The question is important, as two opposite schools of thought exist: that use cases and the scenarios they contain should be as vivid as possible; and that they should be simple, systematic, and generally applicable – i.e. without the character of being examples, specific instances in time and place and person.

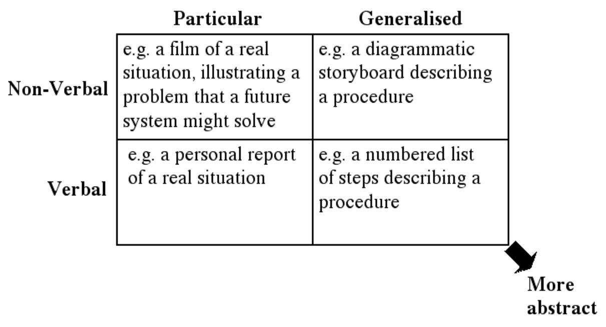
Even from this brief analysis of the question it seems clear that many related types of abstraction are involved when we make scenarios.

Firstly, there is a large group of types of generalisation. These include generalising away from persons (Harry Rogers) and named organisations (The Bank of New Jersey) to classes of actor (the Customer, the Bank). We also generalise away from times and places (at 9:15 am on Tuesday outside the Main Street branch) to abstract situations (anytime, any teller machine). These together serve to discard the vivid and particular verbal description in favour of something clean, short, simple, and reusable. But equally, they create scenarios devoid of many possible clues as to feelings, intentions, and styles of approach that might possibly be important: after all, we are trying to capture people's wishes as requirements, so ignoring much of what they say they want is not necessarily a good idea. Abstraction by generalisation risks 'throwing out the baby with the bathwater'.

Secondly, there are all the types of abstraction implicit in verbalising – and usually also textualising – a description of a real situation or lived episode. In goes a place and time and sunshine and reflections in the glass of the shopping street windows and paving stones and people in the street and trees and a teller machine in the wall of the bank; and out comes … a list of words.

Abstraction to text throws away essentially all the dimensions of sensory experience – sight, sound, touch, smell, taste, acceleration, …; all subjective impressions like surprise, fear, pleasure, complexity, boredom; and all dimensions of space. Even time is reduced to a symbolic indication of sequence, by some device such as numbering or listing. Without becoming too philosophical, we seem to be discarding a dozen dimensions here, replacing them all with a stream of symbols encoded in compressed form with the strange metaphors of natural language. (Think, for example, what images are being referred to when we say 'interrogate the user', 'enter a PIN number', 'capture a requirement' – these are images of violent questioning, going into a house, grabbing a hostage). Text has many virtues, among them familiarity, being a shared medium, expressiveness, flexibility, and brevity; but it is both imprecise and skimpy on detail.

Obviously, abstraction at some stage is essential for making systems which must model some aspects of the world. It seems reasonable that some part of that abstraction should be present in the requirements, with the rest of the abstraction journey coming between the requirements and the final design of the system. But how much abstraction is actually necessary in a specification? Is the right information thrown away at this early stage? How can we know which aspects of experience (if not of reality) are unimportant for a design before the design work has even begun? These are serious and to some extent unanswerable questions, at least in the general case. Each project can, and probably should, decide how abstract its scenarios should be.



	Particular	**Generalised**
Non-Verbal	e.g. a film of a real situation, illustrating a problem that a future system might solve	e.g. a diagrammatic storyboard describing a procedure
Verbal	e.g. a personal report of a real situation	e.g. a numbered list of steps describing a procedure

Figure 1

If that's so, it's rather disturbing that non-verbal means of communicating scenarios – film, tape recordings, photographs, diagrams, sketches, acted scenes, storyboards, … are used so rarely. Similarly, not-totally-generalised text – ethnographic observations, personal reports, actual quotations, tape transcripts, … are also used remarkably rarely. In fact, if vividly 'real' scenarios are ever appropriate, it seems that they are not being used on many projects that would benefit from them.

In conclusion, then, as far as scenarios are concerned, vivid concreteness – as opposed to pale abstraction – can be thought of as consisting of two main axes. Good scenarios are not too verbalised, and not too generalised. How much is too much is a matter for the individual project. We can visualise this on a diagram in the style of a Boston matrix (Figure 1).

It's certainly impossible to make scenarios 'real' – then we'd be into other forms of requirements elicitation, like participative inquiry or work experience – and philosophers from Plato to Kant argue rather convincingly that reality (whatever that is) is inaccessible. In any case, reality is off our diagram, far towards the top and left. But we can and should take some care to choose how much, and what sort of detail to give to the longsuffering readers of our requirements specifications.

RE-Publications

Book Reviews

All books reviewed by Ian Alexander

Problem Frames
Michael Jackson
Addison-Wesley, 2001.
ISBN 0-201-59627-X (paper)

This is in many ways a deeply practical book, but anyone who knows or has read anything by Jackson will know that it must also be razor-sharp on theoretical issues.

One of Jackson's precepts is that one should not invent yet more notations - unless there is very good reason. And especially, one should not propose yet another method for describing problems and solutions - unless all existing methods are inadequate.

Well, this book makes it clear that Jackson believes there are grave weaknesses in current practices. Problems are confounded with solutions; requirements that purport to be about the one are often about the other. The idea that one can write requirements for a system without knowing what it will be - a burglar alarm that is either electronic or a flock of geese - is risible. And the claim that one can identify a point where the requirements are stable and complete is just plain wrong.

In that case we need to think about the problem first, and then, using knowledge of the world and of systems, identify a pattern that accurately relates problem to solution, and conduct analysis and design on that basis. This is not far from what the Object-Oriented community has been claiming for some time, but Jackson does not entirely agree with their prescriptions.

For a start, he does not hold with Use Cases:

The use case view can work quite well when it makes sense to think of the machine as a facility offering discrete services that are used in clearly delimited episodes.

Hmm, yes, Michael, but that is quite a lot of machines actually -- almost anything that provides someone with a screen and a keyboard or pointing device, for a start. And while having clearly delimited episodes is the cleanest case, the keep-on-doing-this-as-long-as-you-need-to sort of problem seems reasonably susceptible to the use case treatment as well. Think of describing a driver's interface to a car - subproblems like steering and stopping are perfectly describable, even though we know they interact and go on more or less continuously. The cases are not 'steer the car' but must consider how to stop the car while steering, so there are more cases than immediately appear. The question is whether the approach is worth applying, or whether a different and perhaps novel approach like problem frames would be better.

In addition, Jackson doubts whether large-scale patterns as posited by the OO 'Gang of Four' often apply:

The ideal use of problem frames is when your whole problem perfectly fits a recognised frame, and the frame provides an effective and systematic method of analysis and solution. It doesn't often happen...

But in some cases, in a highly specialised and refined application area, a large composite frame is developed, and becomes widely accepted...

This is a healthy degree of scepticism, combined with a respect for the other guy's point of view.

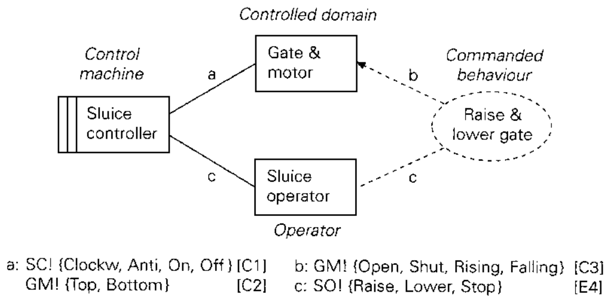


Figure 1. A Problem Frame

In consequence, Jackson believes that we have no choice but to devise a way of dealing with problems and solutions at the same time, refining each as we come to understand more of both. If as is likely no fully-formed pattern can come to our rescue, then the only workable approach is to build up our own analysis pattern from domains and their logical interfaces.

Problem Frames is a lively, vigorous, closely-argued book. It is exceptionally clearly written, always interesting and often provocative. A curious feature is that many of the sections end with an 'Any questions?' subsection. This contains questions that the interested reader might (possibly) ask, immediately answered in conversational if not indeed Socratic style: *'Surely the transformation frame .. should be forbidden.* Your instincts are very admirable!...' This literary device won't suit all readers but the answers do give further information on more advanced topics, enabling the book to be read at different speeds according to the reader's appetite for detail.

It would not be appropriate to venture to answer the 64,000 dollar question, namely whether problem frames will or should take over from Patterns, Objects, and Use Cases as the preferred route from World to Machine. To some extent, choice of technique is governed by fashion; to some extent, by what is familiar. Only when fashion and familiarity let us down do we really start to look for other approaches.

This eagerly-awaited book is as good as everyone thought it would be. Jackson has expanded and systematised his thinking on problem frames, requirements, specifications, architecture, and much else. It is supported by a concise summary of the problem frame and related notations in an appendix, by a glossary that is as carefully crafted as any data dictionary, and by thorough references and indexes. The bibliography does not describe the referenced documents, but there are brief descriptions of the issues in many of the 'Any questions' sections and these do introduce some of the books mentioned.

This book will remain a key reference for anyone thinking about the roots of requirements engineering, its relationship to architecture, and the management of complexity for many years to come. It can be highly recommended to everyone interested in how we can improve our repertoire of techniques and build better systems.

Component-Based Business: Plug and Play
Richard Veryard
Published by: Springer, 2001.
ISBN 1-85233-361-8 (paper)

Richard Veryard is becoming well known to the systems engineering community in the UK as an original and independent thinker. In this book he lives up to expectations of providing a provocative read.

The idea of plug and play is familiar as a concept in personal computer use: you buy a new scanner, or printer or other device, plug it into your computer, and - it works. Under the surface, many strange and complex things may be happening, but, usually, you neither know nor care. Of course, if it doesn't work then you start to care very much.

Most business services simply do not plug and play with each other. What interfaces should such services have? How should they behave? What should they do as the environment changes? Questions like these are Veryard's domain.

The book begins:

We are in the Middle Ages. Knights chase around the forest after mythical beasts, encumbered by heavy armour and ancient weapons. Wizards camp in the clearings, selling silver bullets and magic potions….

We are in the Middle Ages. System engineers try to capture The Business Requirements, to which they can develop an ageless Solution…and customers dream of Frustrations finally Alleviated, Demands Satisfied at last.

Readers may find themselves shifting uncomfortably in their seats at this. How can we move business IT services into the modern world? Does anyone know? Veryard does not claim to have the answers, but he has a wealth of ideas - from management theory, evolutionary biology, computer science, architecture, philosophy, economics, even artificial intelligence.

The style is unusual, almost Socratic. A proposition is stated:

An intelligent system can be composed of unintelligent components.

An example or illustration is then given, often supported by a literary or historic reference - J.S.Mill, Vitruvius, Darwin. It is briefly discussed. Then a question is posed, challenging the reader to become involved:

Q. What's wrong with task specialization? Why doesn't it always lead to economies of scale?

The book ends, indeed, with the author's e-mail adress, and the discussion continues on the book's website, www.component-based-business.com, where you can pose similar questions for debate.

The argument is a blend of erudition, obvious practical experience, and the homespun. The discussion of Availability (a measure of the quality of a service), for instance, includes a definition by Borgmann, a table of ideal 'fantasy' goals and realistic requirements, and illustrations based on what services a business traveller may use from a hotel room. How reliable and universal does a hotel phone/data service have to be for business people to use it? If you arrive expecting to use your own laptop and mobile, will you even discover that the hotel could have helped you?

Q. In practice, how would a traveller learn about a hotel service she wasn't using? Thus the use of a service may be foreclosed by a prior commitment or expectation…

I don't doubt that Veryard will be used as a source of ideas for many years to come. All our disciplines - requirements, systems engineering, project management, and so on - are closely intertwined, just as the businesses in which we work depend on, compete with, predate upon, or parasitize each other. The relationships are complex and shifting. It really doesn't make sense to talk about them as if they were fixed.

There is a short and practical list of references. There really should have been a full index, though this is partially compensated for by the rather detailed Glossary which includes page references as well as Veryard's own definitions of his terms: as he says,

some of [these definitions] are significantly different to those found elsewhere

This is a stimulating, idiosyncratic, quirky book. Consultants, engineers working in business and IT project managers will find it well worth a look - it will give you something to think about while waiting for a plane and may, who knows, influence your thinking.

Effective Requirements Practices
Ralph R. Young
Addison-Wesley, 2001.
ISBN 0-201-70912-0 (paper)

Any book that has Effective in its title and Pragmatic in the cover blurb is likely to be stronger on established practices than on excitingly revolutionary techniques, and this one is no exception. Young treads the familiar ground of the Standish Group's reasons why projects fail, the need for requirements practices that work, the system engineering life-cycle, and a template-and-checklist approach to projects. Novelty and risk are specifically discouraged.

As it happens, copies of Young and of Jackson have arrived at the same time, and the comparison is interesting. Where Jackson looks at the big picture, its theoretical weaknesses, and what should be done about it, Young looks at the specific (large) project, and the approaches most likely to succeed from those available now. Jackson wants you to become a better engineer; Young just wants you to a better job today. Something that they both agree on is that the waterfall model is inadequate: engineering development certainly does not follow a straight line path.

The book is sensibly divided into chapters that directly reflect the ten recommended practices:

1. Commit to the approach.
2. Establish and utilize a Joint Team responsible for the requirements.
3. Define the real customer needs.
4. Use and continually improve a requirements process.
5. Iterate the system requirements and architecture repeatedly.
6. Use a mechanism to maintain project communication.
7. Select familiar methods and maintain a set of work products.
8. Perform requirements verification and validation.
9. Provide an effective mechanism to accommodate requirements changes.
10. Perform the development effort using known, familiar proven industry, organizational, and project best practices.

The final chapter is entitled 'How to Proceed' and contains a list of key issues, a five-page checklist for the requirements-related activities on your project, and tips on prioritization and the CMM.

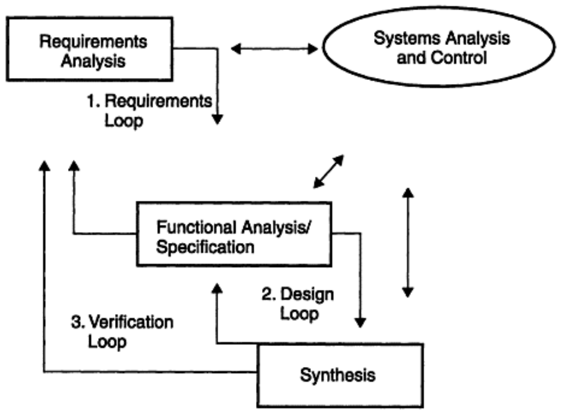


Figure 1. Trade-off Loops in the System Engineering Life-Cycle

Each practice is described in about twenty pages (twice that for defining the *real* customer needs - a wise emphasis on a crucial topic; techniques mentioned approvingly include workshops, use cases, and prototyping). Each chapter begins with a short summary. Then it introduces its key themes and defines its terms. Young is not afraid to draw on best practice, whether described by industrial colleagues or in market or academic research. For example, on Joint Teams he quotes Telelogic's Writing Better Requirements course approvingly, borrowing the illustration of Customer and Supplier Roles Throughout the System Life Cycle - a diagram that makes clear that requirements are not something that happens at the start of a project and then disappear. The chapter ends with a 200-word summary and a couple of pages of key references and suggested readings. Each book mentioned is described in a paragraph of descriptive praise, giving the reader a clear idea of what they would get if they purchased and read it.

This is a book "dedicated to the systems and software engineering professionals who have devoted their life's work to serving others" - a daunting sentiment. Young braves a blizzard of acronyms to guide the reader in a plain and readable style through CMM and MOEs and TQM, though his rendering of SWAT Team as 'strategic weapons attack team' is perhaps a little too close to Armageddon: Special Weapons and Tactics (police) Team may be what was intended. Young is always clear and practical, and provides plenty of supporting evidence for claims and suggested approaches.

The book has up-to-date lists of tool vendors, good books on each of the ten practices, and efficient indexes. It is solidly aimed at decision-making practitioners, and given the stress on systems engineering this means both project managers and chief engineers on large system projects. As the author says,

It is my sincere hope that one use of this book will be to provide practitioners [with] the experience and data to encourage P[roject] M[anager]s to provide adequate funding for requirements-related activities... This is obviously an issue that needs to be addressed in corporate and organizational training programs for PMs.

Amen to that, though there is only so much experience one can gain from reading books.

Such people will find this book plain, practical, and useful, and they may well benefit from the simple artefacts on the supplied CD. Students of software engineering may find the book too industrial, and they certainly won't find details of the latest and most exotic techniques here. Students of systems engineering will find it a worthwhile and systematic overview, though without exercises. In summary, this is a solid guide to the requirements aspects of systems engineering.

RE-Sources

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

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

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

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

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

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

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

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

Reduced rates are available to all RESG members when subscribing to the REJ. Special 2001 Personal Rate £45.00 (A saving of £19.00).

Mailing lists

The SRE list

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

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

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

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

RE-Actors

The committee of RESG

Patron: Prof. Michael Jackson, Independent Consultant. E-Mail: jackson@acm.org.

Chair: Prof. Bashar Nuseibeh, Computing Department, Faculty of Maths and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-Mail: B.A.Nuseibeh@open.ac.uk, Tel: 01908 655185