It's all about Risk, Mr Glinz

Ian Alexander[1]

[1]Scenario Plus, London, England
iany@scenarioplus.org.uk

Abstract. Requirements Engineering (RE) often proceeds with little mention of risk. RE discourse instead covers topics such as stakeholders, goals, scenarios, qualities and measurements (acceptance criteria). Only in specialisms such as safety and security engineering is risk commonly analysed, in the form of hazards and threats respectively. Yet all of RE, indeed all of Systems and Software Engineering, exists only to mitigate project risk. This paper revisits the reasons why that is so, unpacks ways that risk can be treated, lists ways that RE handles risk, and reflects on lessons from this analysis.

Keywords: Requirements Engineering, Risk, Traceability

1. Introduction

After a talk on the elements of requirements that I gave at REFSQ some years ago, Martin Glinz asked why I had not mentioned risk. To be strictly fair, I had briefly mentioned Risk Models, as one of at least 8 ways to document rationale [1]. But more importantly, I had taken in with my mother's milk as a DOORS novice that RE was all about risk: that everything we did was there to control and reduce risk; it was the purpose of the whole of Systems Engineering (SE), indeed; it was what we taught people in the first section of any course on RE or SE. Everyone in industry was well aware that projects could and regularly did fail. Back in 1995, the Standish Group had written their infamous "Chaos Report" [2]. Richard Stevens, the inventor of the DOORS requirements traceability tool, had insisted on starting our book with a table based on Standish giving the reasons for project failure [3]. In short, I hadn't talked about it because it seemed too obvious. It was part of the woodwork, not an item on the workbench to be examined and discussed. It was the invisible cultural element that everybody that I worked with "knew" but never talked about. As Edgar Schein says, culture consists of the shared, unspoken assumptions held by a group and learnt through experience. [4]

With all this tacit knowledge hanging over me, I mumbled that I had mentioned risk; stammered something about RE being all about risk, and that of course you could model it; and stopped. Of course, I should have asked Martin to say how he thought risk fitted into the bigger picture of requirements. But then, he has written and spoken about risk on various occasions. [5][6]

The lesson, naturally, is that when Martin asks a question, it's bound to be a penetrating one. What is the relationship between risk and requirements? How should risk be represented in a requirements model, in a project's documentation? Do we actually understand in a practical way how to deal with risk in all its forms, or is our knowledge hidden away in an unread Risk Register? How, in fact, do theory and practice join up around risk, if at all? What would be the "staircase wit" [7] reply to Martin's question?

Requirements exist to reduce development risk, or as Martin rightly says, the risk of not meeting the stakeholders' desires and needs. They therefore add value

to a project if the benefit of risk reduction exceeds their cost. [6] Further, Martin adds, we can assess the risk involved in an individual requirement based on its criticality and other factors – do we understand it, does this system need to obtain a safety certificate, and so on – and then we can make a triage decision. If the stakeholder is important or the impact of a failure on this requirement will be high, then it deserves high effort. [6] That Glinzian answer implies that every requirement should trace to a stakeholder – much as Richard Stevens used to say that every user requirement should begin with a named user role. [8] The answer also paradoxically implies that while risk is pervasive in RE, it should be treated implicitly: applying high effort and developing traceability are not overtly about risk.

This paper looks at how RE addresses risk (section 2); how risks are made explicit in Systems Engineering in general (section 3); how risks can be modelled explicitly in RE (section 4); how RE typically handles risks implicitly, via traceability (section 5). The paper concludes with some brief reflections on requirements and risk (section 6).

2. Addressing Risk

How does RE, or for that matter systems or software engineering in general, act to reduce risk? Is it just a totem, an amulet, or are there reasons to believe it actually works?

Consider a project that does not make any special effort to discover and trace its requirements. People intuit in a general sort of way what needs to be done. Engineers make drawings. Programmers write code. User interface designers sketch screens, design consoles. Workers cut metal. Electronics wizards make printed circuit boards. Eventually there's something that looks like a system. A user tries to use it. Of course, it fails here, there and everywhere. Now, at what should be the quiet moment of handover, all kinds of "integration" problems arise: the subsystems do not fit together. Testing is, in fact, discovering missed requirements at all levels: component, subsystem, system, even user – for when users experience a new system for the first time, they immediately notice features that they would like to add. Supposedly final delivery becomes first prototype, first iteration. This can be extremely expensive, at least on large projects.

Firstly, all the work done on the basis of the misunderstood, undocumented, or undiscovered requirements must be redone, so the project has conducted nugatory work.

Secondly, project teams grow with time. By the stage of actually manufacturing hardware, or of detailed design and coding, every day on the project costs many salaries: there is a large "marching army" cost to doing anything. It is therefore an extremely expensive time to make any changes, let alone to start again on all the requirements.

Therefore, projects are wise to select a suitable development life-cycle, with carefully-orchestrated phases specifically designed to reduce risk. Activities are selected to control the risks associated with the project's situation. For example, if there is a large technological risk because an organisation is trying a new technology for the first time, feasibility prototyping and development of technology demonstrators is justified. [9] These activities, these life-cycles, have the explicit purpose of discovering problems early, when they can be resolved cheaply – if need be by cancelling the project, but more likely by selecting design options that can be seen to work and are known to satisfy stakeholders' goals.

If we assume for the sake of argument that a given project has a certain number of problems to overcome, that project can be imagined to have two choices. It can leave things to chance, in which case it will discover the problems late. Or it can conduct risk-reducing, requirement-discovering activities early in the life-cycle, in which case it will discover many of the problems early. Since costs rise steeply with phase in the so-called 1:10:100 rule – perhaps 1 unit of work is needed to discover requirements, 10 to design from it, and 100 for code,

manufacture, and test [10] – the early approach is much less costly. Since risks such as of embarrassing project and public relations disaster similarly rise steeply when problems occur late and large, the early approach is also much less risky. In project planning jargon, where time is shown as the X-axis on charts and early is on the left, systems engineering offers a welcome "left-shift" in problem discovery and resolution. Hence, large projects should, amongst other things, engineer their requirements.

Not everybody believes this. Small projects may escape the logic. And traditionally, many systems were produced with few explicit requirements – design drawings were enough. The 1956 model of car was the same as the 1955 model, with somewhat larger tail fins, somewhat more chrome plating, somewhat more exaggerated front and rear bumpers, somewhat more sculpted curves of the bodywork, a somewhat more finely tuned engine. This was Modification Engineering. It works very well – up to a point. It is unfortunately impossible to predict when that point is reached, when complexity suddenly tips projects into chaos. One project succeeds: the complexity proved containable. Another, that didn't look so different, so much more complicated, fails catastrophically. Perhaps the elegant mathematics of chaos theory might be applied to project trajectories. At any rate, such variation in project outcome may explain why people are able to hold opposing beliefs about the need for requirements, and the underlying risks that they mitigate.

3. Making Risks Explicit

Risks can be handled in many different ways, generally by making them more explicit. It is notable that methods for discovering risks are often used by other disciplines than RE.

Project managers typically maintain a Risk Register, a list of known risks that could derail the project. The response is to identify and carry out appropriate risk mitigation actions on the project, as well as to monitor the world for events outside the project's control. Requirements risks are only one of several possible types that may afflict projects. Ian Sommerville, for example, lists technology risks, people risks, organisational risks, tools risks, requirements risks, and estimation risks. [11] This list, good as it is, itself carries with it the danger that the headings will divert attention away from other important risks that do not fit into any of these types. For instance, a product development project may succeed in creating a good working product, on time and to budget, but the product could fail on the market if a rival product comes to market earlier, or if market conditions change for the worse and consumers fail to buy the product. Managers need to be aware of such risks and to be ready to replan should the triggering events, which James Dewar calls "signposts", in fact occur. [12]

Safety engineers conduct a hazard identification process, often based on a list of known hazards, that is, things that specifically endanger safety. Hazards were historically often discovered when systems failed catastrophically, leading to death, injury, or destruction of property. The response was to extend the list of known hazards related to specific design elements, and to write safety standards – reusable quality requirements, often enshrined in law – to enforce safer systems. [13] For example, if you have a steam boiler which may explode, you must have boiler-specific safety requirements, now standardised, and enforced by government inspectors. Thus the BSI has a standard which "applies to water-tube boilers with volumes in excess of two litres for the generation of steam, and/or hot water at an allowable pressure greater than 0,5 bar and with a temperature in excess of 110 °C as well as auxiliary installations (other plant equipment)." [14] You couldn't have a clearer case of requirements coming from design, rather than the other way around.

Security engineers similarly base their requirements on threat analysis, often based on lists of known threats. These too, unfortunately, were often discovered the hard way. The reason is simple enough: while the goal for security is simple (be secure), achieving it is anything but. As with safety, the necessary

requirements depend intimately on the details of the design. For example, if you have a connection to the internet, which may bring hackers, trojans and viruses, you need specific mechanisms such as a firewall and an anti-virus tool. The presence of risk here is unmistakable: ticking all the boxes on the security counter-measures list is no guarantee there will not be a break-in.

A complementary approach is to employ a "tiger team" with the explicit task of trying to break the security of the system under design. As Ian Sommerville says, "They simulate attacks on the system and use their ingenuity to discover new ways to compromise the system security". [15] This is an interestingly Glinzian approach: if you can't quantify your security formally, you can at least try it out operationally. Perhaps it's what Martin would have written in [5] if security had been his focus.

Systems engineers may, analogously, conduct a risk identification process. Responses can include continuing with the existing approach (risk acceptance); prototyping (risk exploration); changing the system and its specifications (risk mitigation, or perhaps risk prevention); and cancellation (risk avoidance). [16]

Clearly this list could be extended to human factors and other areas.

It is immediately apparent that by no means all the risks that project managers, safety engineers, security engineers and systems engineers can discover trace to users; just as by no means all system requirements trace to user requirements. Some, as we have just noticed, will trace to standards which enforce rules such as safety and security; others may simply be seen to be necessary during design. Requirements may also be driven by usability standards; rules limiting interference, like the radio regulations; and financial probity rules, which heavily influence software for banking and insurance. One can argue that requirements and matching risks of these kinds can be traced to stakeholders of type Regulator [17]; but if engineers identify a new risk, they still want to handle it appropriately, whether a regulator is involved or not.

4. Modelling Risks

The simplest kind of risk model is simply a list, which at least presents each risk openly as a traceable (numbered) item, and invites action on each item, with verification through traceability.

The next logical step is to prioritise the list. Barry Boehm, for example, suggests identifying a top ten [18]; Sommerville observes that this might be the wrong number for any particular project. [11]

A weakness of using a simple list of risks, prioritised or not, is that risks may not be independent, in which case it makes no sense to deal with them separately, one by one. Requirements analysis can model dependencies, as for chains of goals in i* [19] or for claims, arguments, and evidence (or simply assumptions) in safety and other rationale models [20] [21]. A goal modelling approach which essentially models threats and mitigations is misuse case analysis, which lends itself to simple graphical models. [22] [23]

As with safety and security requirements, which are in practice handled not by requirements generalists but by safety or security engineers, so risk management has become a speciality of its own, with good books written by experts like Martyn Ould and Capers Jones. [24] [25] We may approve specialisation or deplore fragmentation, according to taste, but the effect is perhaps to remove overt consideration of risk from the world of engineering of requirements, software and systems to the world of experts in business risk, to the extent that the topic itself is unfamiliar to engineers. Perhaps, then, it may be worth reflecting on ways that risk can be handled in the context of requirements. Several such ways have already been mentioned more or less directly above.

5. Handling Risks with Requirements

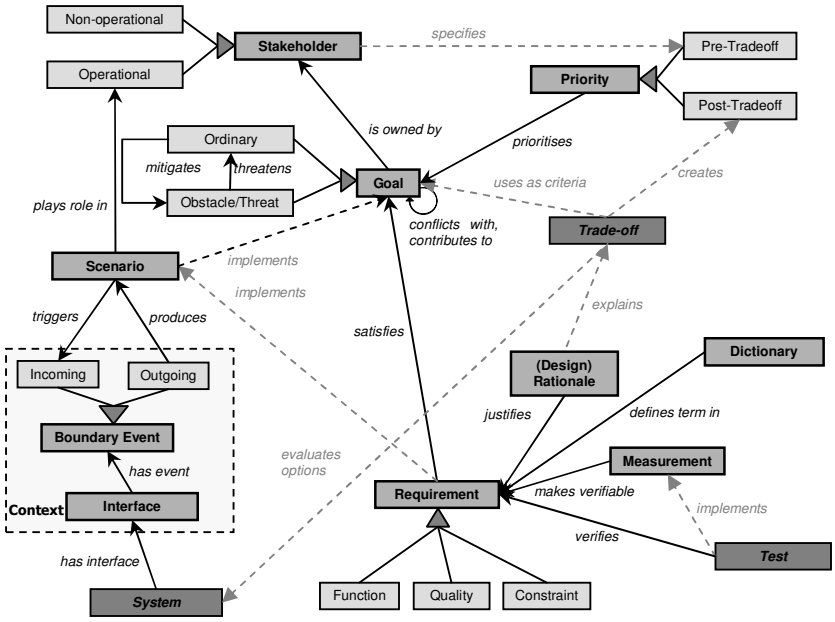

Fig. 1. A Requirements Metamodel.
Traces are shown as solid lines, logical relationships as dashed lines.
Risk can be shown explicitly in Rationale, or as Threat; implicitly, it is pervasive.

It may be as well to repeat, here, that all requirements work is done to manage risk. The list of specific techniques suggested here consists, therefore, of examples of risk management approaches using requirements. The examples are taken from the requirements metamodel of [26] (Figure 1).
Note: The word "trace" as used below can be interpreted either literally, meaning actual database links between separate database records, or conceptually, meaning for example that a field is added to a table to include the "traced" facts, or simply that a reference is added in a document to indicate that the facts are connected.

Risk of not discovering a key stakeholder: a systematic search can be made for possibly-involved people and organisations, whether these are involved in day-to-day operations with the system or not.

Risk of not attending adequately to stakeholders: user-derived requirements can explicitly be traced back to their stakeholders, who in turn can be analysed with an influence matrix. [27]

Risk of paying attention to the problems you can solve, rather than the ones that stakeholders want solved: Verifiable requirements can trace back to not-necessarily-deliverable stakeholder goals.

Risk that system scope is not fully understood: model system context; validate context with stakeholders; trace scenarios to context elements (in and out interfaces); trace interfaces to design.

Risk that goals cannot all be delivered together: use traces to show goal conflicts as well as goal contributions; model obstacles and threats as well as positive goals, and show the threatens/mitigates relationships between these.

Risk that not all the important requirements can be delivered (on time, to budget): prioritise and trade-off the goals with the available solutions (design options).

Risk that requirements are gold-plated: prioritise; use traces to design to evaluate cost per requirement (impact analysis); use traces to stakeholder goals to see why each requirement must exist; use traces to scenarios (use cases, stories) to see how each requirement will be used; validate scenarios with stakeholders.

Risk that stakeholders and project members interpret terms differently: trace terms used in requirements to project dictionary; validate dictionary with stakeholders.

Risk that requirements are deleted to save time/money: trace to rationale to show why each requirement is needed in the system design, and why each trade-off decision was made.

Risk that requirements are not fully and correctly implemented: trace to tests to ensure coverage; make each requirement atomic (separately testable as a single item); write measurable acceptance criteria for each requirement; use traces to scenarios to show how the requirements will work.

Risk of not understanding complex risks: make an actual risk model to show the dependencies between risks, and if need be their quantitative relationships; trace risks in the model to requirements. Much the same applies to specialised approaches for safety, security, reliability and other system qualities (non-functional requirements).

Again, this list could be extended indefinitely, covering different authors' metamodels and every requirements technique ever invented.

6. Reflections on Requirements and Risk

Some simple but striking observations can be made from this analysis.

Risk is pervasive in requirements engineering: every model is created to reduce project risk.

There is no separation of types of risk in the world; any type of risk – commercial, schedule, technical, stakeholder, safety, regulatory, market, … – can impinge on a project. Focussing on just some well-understood types of risk is itself risky.

There is no guarantee that anybody's checklist or risk procedure is complete.

Traceability is the primary mechanism of requirements engineering, and the primary purpose of any worthwhile requirements tool. Traceability is accordingly the means by which requirements control risk.

Complex traceability patterns are probably best represented graphically, in forms such as goal and rationale models. But this remains quite rare in industry, at least outside safety argumentation, where the need to describe complex chains of reasoning about handling risk is greatest.

It's ***all*** about risk, Mr Glinz.

Bibliography

DR: Alexander, Ian; Beus-Dukic, Ljerka. *Discovering Requirements: How to Specify Products and Services*. Wiley, 2009.

SSUC: Alexander, Ian and Maiden, Neil. *Scenarios, Stories, Use Cases through the systems development life-cycle*. Wiley, 2004.

WBR: Alexander, Ian and Stevens, Richard. *Writing Better Requirements*. Addison-Wesley, 2002.

References

[1] Alexander, Ian. *Piecing Together the Requirements Jigsaw-Puzzle*, Keynote Talk. Slide 30, "Ways to document rationale". in Roel Wieringa, Anne Persson (Eds.): REFSQ

2010, Essen, Germany, June 30 - July 2, 2010. Proceedings. Lecture Notes in Computer Science 6182 Springer 2010. Available at http://www.scenarioplus.org.uk/papers/reqts_jigsaw/reqts_jigsaw.pdf

[2] Standish Group. *The Chaos Report*. 1995. http://www.standishgroup.com

[3] **WBR,** Table 1.1, "Reasons for project failure".

[4] Schein, Edgar H. *Organizational culture and leadership* (4th Edition). Jossey-Bass. 2010.

[5] Glinz, Martin. *A Risk-Based, Value-Oriented Approach to Quality Requirements*. IEEE Software, 25 (2). Pages 34-41. 2008.

[6] Glinz, Martin. *Quality Requirements – A New Look at an Old Problem. Keynote Talk for RE 2012.* http://2012.reconf.de/fileadmin/PDF_Dateien/REConf_2012/Vortraege/Keynote_-_Dienstag_-_TK2_-_Keynote_Glinz_REConf2012_106322369.pdf

[7] Diderot, Denis. *Paradoxe sur le comédien*, 1773, remanié en 1778; Diderot II, Classiques Larousse 1934, p. 56 "l'esprit de l'escalier" (thinking of the perfect answer too late, after you've left the salon, and are walking down the stairs). The elegant loan-word *treppenwitz* is used in English, but unfortunately it has a different meaning in German.

[8] **WBR,** "Anatomy of a good requirement", pages 97-98.

[9] **SSUC,** Farncombe, Andrew. "Project Stories: Combining Life-Cycle Process Models". Pages 299-324.

[10] Rice Consulting. *The Economics of Testing*. http://www.riceconsulting.com/public_pdf/STBC-WM.pdf "The general rule is 1:10:100" naming the three stages as requirements and design, system/acceptance testing, and production. Other sources give the stages as analysis, design, and code production.

[11] Sommerville, Ian. *Software Engineering*. Addison-Wesley, 6th Edition, 2001. "Risk analysis", pages 86-88.

[12] Dewar, James A. *Assumption-Based Planning: A Tool for Reducing Avoidable Surprises.* Cambridge University Press, 2002.

[13] Lutz, Robyn R. "Software Engineering for Safety: A Roadmap". In *The Future of Software Engineering*, Anthony Finkelstein (editor), ACM Press, 2000. http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finallutz.pdf

[14] British Standards Institution (BSI). *BS EN 12952 Water-tube boilers standards.* http://shop.bsigroup.com/en/Browse-by-Sector/Manufacturing1/Pressure--equipment/Water-tube-boilers/BS-EN-12952-Water-tube-boilers-standards

[15] Sommerville, Ian. *Software Engineering*. Addison-Wesley, 6th Edition, 2001. "Security assessment", page 483.

[16] Stevens, Richard; Brook, Peter; Jackson, Ken; Arnold, Stuart. *Systems Engineering: Coping with Complexity*. Prentice Hall, 1998. "Decisions and risks", pages 164-168.

[17] **DR,** "Regulators", pages 32-33.

[18] Boehm, Barry. *A Spiral Model of Software Development and Enhancement*. IEEE Computer, 21 (5), May 1988, pages 61-72.

[19] i*, an agent- and goal-oriented modelling framework. http://www.cs.toronto.edu/km/istar/

[20] **DR,** "Documenting Rationale", pages 169-183.

[21] Burge, Janet E.; Carroll, John M.; McCall, Raymond; Mistrik, Ivan. *Rationale-Based Software Engineering*. Springer, 2008.

[22] Alexander, Ian. *Misuse Cases: Use Cases with Hostile Intent*. IEEE Software, 20, (1), Jan-Feb 2003, pages 58-66.

[23] Sindre, Guttorm; Opdahl, Andreas L. *Eliciting Security Requirements by Misuse Cases*, Proceedings TOOLS Pacific 2000, 20-23 November 2000. pages 120-131.

[24] Ould, Martyn. *Managing Software Quality and Business Risk*. Wiley, 1999.

[25] Jones, Capers. *Assessment and Control of Software Risks*. Prentice Hall, 1994.

[26] Alexander, Ian. Model-based Requirements Discovery. RuSEC, Moscow, 23-24 September 2010. http://rise-russia.org/rusec2010

[27] **DR,** "Prioritising Stakeholders", pages 43-44.