Certainty, Risk and Gambling in the Development of Complex Systems

Andrew C Pickard, Andrew J Nolan and Richard Beasley
Rolls-Royce plc
Andrew.C.Pickard@rolls-royce.com

Abstract.

One of the few things certain about development of complex systems is the requirements for the system will remain uncertain late into the development program. Even though we recognize this, why do many programs assume that requirements will not change (gambling) or treat requirement change as a risk rather than a certainty? An analysis of gas turbine engine control systems requirements shows that typically 50% will change between Critical Design Review and Entry into Service. This requirements uncertainty manifests as technical risk to the programme. This paper evaluates the impact of not managing these uncertainties and describes how applying Systems Engineering principles can reduce this effect.

Design iterations produce rework, and without technical risk management, ~50% of the effort will be wasted in producing unacceptable designs. In turn, correction of these errors will create more rework. This results in much iteration and cumulatively twice as much work is required to produce a mature design. With technical risk management, less than 10% of the design effort will result in rework, giving a mature product with far less iteration.

Investigation into the source of the product development lifecycle problems indicates many escapes occur during requirements definition and review. Around twice as many changes are driven by errors in requirement definition by the developer, compared to customer-driven requirements changes. In addition, many of these escapes are detected later in the product development lifecycle than they could have been found, resulting in further escalation of the cost of product development. This paper compares and contrasts the root causes for these escapes during system, software and hardware development and looks at the differences in consequences.

The role of the Systems Engineer is critical in addressing these problems. In addition to effective elicitation of requirements from stakeholders, the Systems Engineer must manage volatility and apply robust design principles to protect against the impact of requirements changes. The Systems Engineer must flow requirements effectively from system to sub-system, from sub-systems to components and provide the "glue" that results in the integrated system being more than the sum of its parts.

Some tools and techniques are presented to help Systems Engineers identify probable sources of uncertainty and provide effective mitigations. An approach is also presented for assessing the technical risk management "maturity" of a project, based upon ‘best practice’ approaches taken from those projects achieving low levels of engineering scrap and rework during their development phase.

Introduction

Rolls-Royce

Rolls-Royce provides power systems and services for use on land, at sea and in the air, and operates in four global markets - civil aerospace, defence aerospace, marine and energy. Predominantly, but by no means exclusively, the business is based around the gas turbine. The points made in this paper apply to the whole range of Rolls-Royce products.

Products and business environment

The power systems that Rolls-Royce produces are complex, and use many novel and cutting edge technologies. Therefore product development is potentially high risk. The nature of the systems we produce means there is a strong probability of the emergence of unwanted properties or attributes, and the need for expensive rework to remedy the undesired system behaviour.

In all the business sectors where Rolls-Royce operates there are demands for increased capability of the power systems, cheaper and faster product development, better transition to operation and better in-service cost and availability. For over 50% of our business Rolls-Royce has responsibility for cost of ownership and availability.

All of the emerging pressures on product development mean that rework is even less acceptable than it was in the past, and the commercial consequences are greater. Rolls-Royce is applying Systems Thinking to reduce rework.

Effect of Rework

It is important to realise that scrap rates apply to the rework as much as the original design, resulting in a cumulate scrap due to repeat rework on some items. Figure 1 shows the relationship between scrap rate and cumulative scrap. A 50% scrap rates implies 100% overspend. However 10% scrap rates implies only 11% cumulative overspend. There is a similar impact on the number of iterations to achieve a mature system, and hence the time to develop the system. This paper looks at the drivers of scrap and rework - and how these can be controlled to achieve acceptably low levels of scrap and rework.

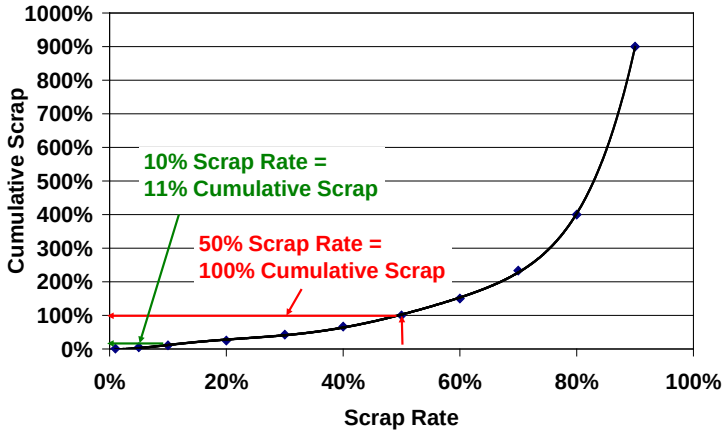

Figure 1. The Cumulative Effect of Scrap

Why this is relevant to Rolls-Royce (and others)

One of the "usual suspects" as a driver of scrap and rework is late change in customer requirements. Customer requirement changes do have an impact, but analysis of the source of rework for engine control systems shows only 16% of change is driven by the customer, 17% comes from internally derived improvements and new requirements, the rest comes from internal

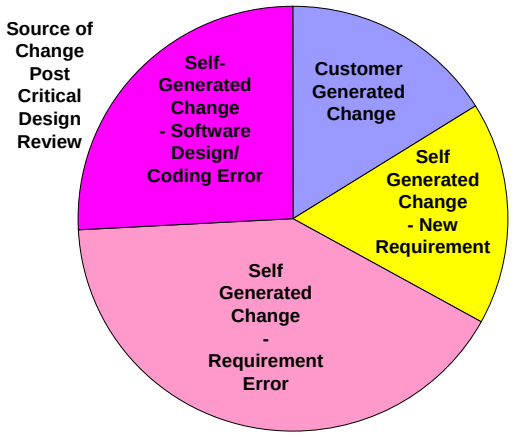

Figure 2. Most Uncertainty is Self-Generated

errors in requirements and implementation, as shown in figure 2. There is a large opportunity to reduce development program costs and achieve maturity earlier, based on three elements:

- Reduce the number of problems created in the first place - reduce the amount of scrap and rework "churn" in the program
- Find problems and address them at the correct point in the system development lifecycle.
- Gain a better understanding of Customer requirements and improvement opportunities to reduce the level of change after program launch, for instance by using techniques like Continuous Early Validation (CEaVa) (1, 2)

Figure 3 shows a typical distribution of where problems were found, and where problems should have been found, during development of software for an engine control system. In this case, nearly 70% of the problems were detected at the correct point in the lifecycle, but the 30% that escaped more than doubled the cost of correcting the problems, using a standard lifecycle cost model for software (3).

Software Problem Report Analysis Should have been found during: --> Found during:	Matlab Animating	Reviewing	Application S/W Building	Low Level Testing	S/W Verification Testing	HSI Testing	System Verification Testing	Hardware Rig Testing	Engine Testing	Airframe Testing	Flight Testing	In Service	Key:	Cost Weight	Cost if found at right stage	Actual cost
Matlab Animating	0.5%												>= 8%	1	0.032	0.005
Reviewing	1.3%	55%											4% to 8%	1	0.798	0.566
Application S/W Building		0.7%	1.4%										2% to 4%	1	0.014	0.021
Low Level Testing	0.2%	2.6%		0.8%									1% to 2%	1	0.012	0.035
S/W Verification Testing		0.8%			0.4%								< 1%	5	0.031	0.060
H/W - S/W Integration Testing		1.0%				0.5%								5	0.080	0.077
System Verification Testing	0.9%	9.9%		0.4%	0.2%	0.8%	5.0%							25	1.592	4.301
Hardware Rig Testing	0.0%	1.3%					0.1%	0.4%						50	0.376	0.907
Engine Testing	0.1%	1.6%		0.0%		0.2%	0.4%	0.2%	1.6%					50	0.885	2.057
Airframe Testing	0.0%	3.8%				0.1%	0.6%			1.2%				50	0.796	2.875
Flight Testing	0.0%	2.3%		0.0%			0.3%	0.1%	0.1%	0.4%	1.5%			50	0.774	2.433
In Service		0.5%					0.0%	0.0%			0.0%	0.1%		200	0.177	1.415
Total Escapes	2.7%	25%	0.0%	0.4%	0.2%	1.1%	1.4%	0.4%	0.1%	0.4%	0.0%		31.2%	Total:	5.567	14.752
Total	3.2%	80%	1.4%	1.2%	0.6%	1.6%	6.4%	0.8%	1.8%	1.6%	1.5%	0.1%	100.0%	Cost Ratio:		265%

Figure 3. Where Found/Where Should Have Been Found, Software Problems

Figure 4 shows similar information for various hardware problem reports covering a range of components in gas turbine engines. In this case, around 60% of the problems are found at the correct point in the development lifecycle. However, the pictures are similar in that most problems are being found during requirements or design review, and most escapes are also in this area.

Gas Turbine Hardware Problem Report Analysis Should Have Been Found During: → ↓ Found During:	1 - Requirements Validation Activity	2 - Requirements Review	3 - Design Review	4 - Manufacture	5 - Component Test	6 - Module Assembly	7 - Module Test	8 - Development Engine Assembly	9 - Engine Development Test	10 - First Article Inspection	11 - Engine Certification Test	12 - Production Engine Assembly	13 - Production Engine Test	14 - Flight Test	15 - In Service	16 - Other	Key:
1 - Requirements Validation Activity																	>= 8%
2 - Requirements Review		33.9%															4% to 8%
3 - Design Review		1.1%	2.2%														2% to 4%
4 - Manufacture	0.4%		0.7%	0.7%													1% to 2%
5 - Component Test		1.1%	1.5%		3.3%												< 1%
6 - Module Assembly			1.5%		0.4%												
7 - Module Test			0.4%														
8 - Development Engine Assembly			1.8%														
9 - Engine Development Test		3.3%	9.1%	0.4%	1.5%				13.9%								
10 - First Article Inspection																	
11 - Engine Certification Test									0.4%								
12 - Production Engine Assembly																	
13 - Production Engine Test																	
14 - Flight Test			0.7%						0.4%			0.4%					
15 - In Service	0.4%		6.9%	2.2%	1.1%				3.3%		0.7%	0.4%		0.4%	1.5%		
16 - Other			0.4%	0.4%												3.6%	
Total Escapes	**0.7%**	**5%**	**23%**	**2.9%**	**2.9%**	**0.0%**	**0.0%**	**0.0%**	**4%**	**0.0%**	**0.7%**	**0.7%**	**0.0%**	**0.4%**	**0.0%**		40.9%
Total All	**0.7%**	**39%**	**25%**	**3.6%**	**6.2%**	**0.0%**	**0.0%**	**0.0%**	**18%**	**0.0%**	**0.7%**	**0.7%**	**0.0%**	**0.4%**	**1.5%**	**3.6%**	100.0%

Figure 4. Where Found/Where Should Have Been Found, Hardware Problems

Certainty, Risk and Gambling

Uncertainty is certain. 2 years research into how to manage uncertainty has shown that the majority of uncertainty is relatively easy to identify, define, quantify, document and manage. The results can be spectacular with savings of 100:1 return on investment!

Unwarranted certainty

The typical “can do” culture has a negative attitude towards uncertainty – it is better to be certain and wrong then uncertain and right! However, the evidence is that uncertainty is very normal.

If not managed, uncertainty will manifest later in a Project's life as late change and rework. The problem begins with the wrong attitude towards uncertainty. One approach is to change the emphasis at gate reviews, so that a project must show where it is certain, where it is still uncertain and a plan for how to address the **residual uncertainty.**

Gambling

Many projects claim they do good risk management but there is little evidence of much activity beyond project (cost and schedule) risk identification. Investigation of one project showed 50% of people claimed they identified Technical Risks but only around 10% actually did anything with the results!

Taking on risk and doing nothing about it is no better than gambling.

Risk management – the middle path

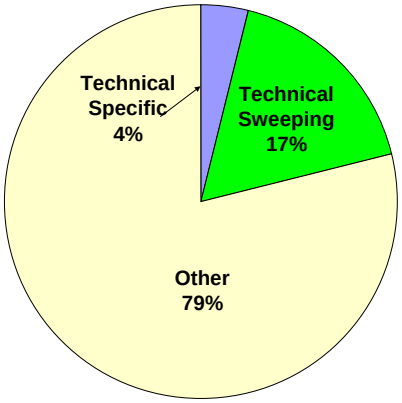

Figure 5 Risk Categories

A review of project risk logs is shown in figure 5.

“Technical Specific” issues relate to a risk applicable to a single function or component. These are very targeted and well defined risks.

“Technical Sweeping" issues are broad brush, technical issues. These issues tend to be closed as “acceptable risk” and appear on every project risk log. . Many of these are not risks – they are certainties.

“Other" issues are non technical risks affecting cost and schedule. These risks tend to be populated by the project managers rather than the technical specialists.

Risk Management tends to be a project management tool, referred to more in the project management processes and guidelines than the engineering processes.

What is Technical Risk Management?

Technical Risk Management is the method by which product (technical) uncertainty can be defined and mitigated. Best practice from across Rolls-Royce and Industry, shows that Technical Risk Management is an effective tool to define, manage and mitigate uncertainty. Mawby and Stupples (4) state:

Track record shows that the vast majority of complex projects are going to overrun their cost and schedule targets, often by large margins. These overruns are usually caused by rework that has been generated within the project by its inability to manage the inherent uncertainty

Technical Risk Management compliments standard Risk Management by focusing on the product and any uncertainties in its purpose, requirements, definition, development and

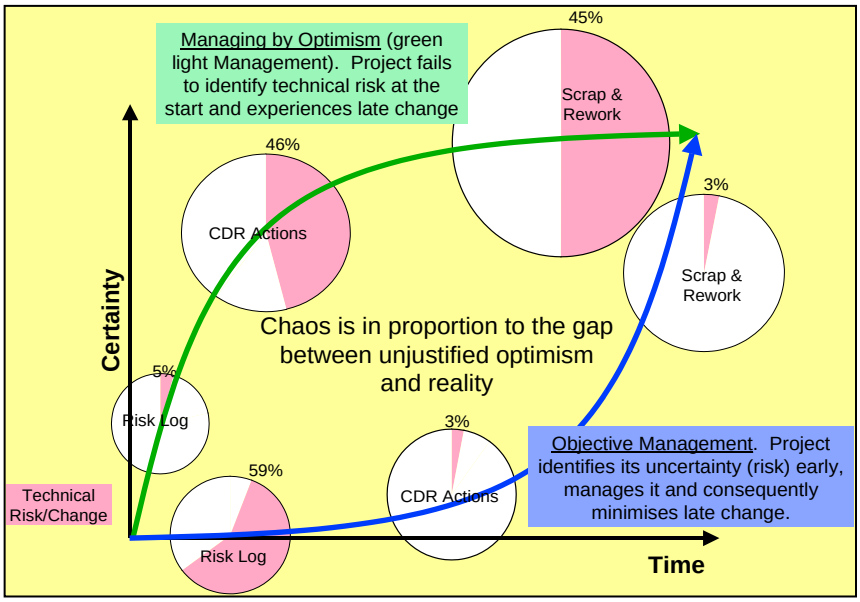

Figure 6 Perception and Reality

deployment.

There are similarities and differences between traditional Project Risk Management and Technical Risk Management. The common issues are planning, process, level of rigor, allocation of resource, and risk criteria.

Planning only on the basis of success and not accommodating risks is likely to lead to failure (5). Such plans are often referred to as “Green Light Plans”.

Figure 6 shows the difference between actual uncertainty (Blue Line) and the level of uncertainty that is often perceived (Green Line) which is unrealistic optimism. A project run on a Green Line tends to report few risks but the risks start to emerge at gate reviews and later on as scrap & rework.

Late change is a function of the gap between perceived and actual certainty - DeMarco and Lister (6) state

“Risk management is Project Management for adults - considering only the rosy scenario and building it into the project plan is real kid stuff."

A “Blue Line” project will identify the risks early but few new risks will be identified during gate reviews and there will be lower scrap & rework.

An example of Technical Risk Management at work

	Knowns	Unknowns
Known	Known-Knowns (28% of uncertainty arises here) *We failed on implementation*	Known- Unknowns (29% of uncertainty arises here) *We know we have risk*
Unknown	Unknown Knowns (30% of uncertainty arises here) *We knew but forgot*	Unknown Unknowns (13% of uncertainty arises here) *Surprises*

Table 1. Knowns and Unknowns

Uncertainty can be managed – but it needs to be recognized. There is the well established known and unknown 4 quadrant matrix. Table 1 contains the root cause distributions from the analyzed projects.

Technical Risk Management is the process of moving risks from where they are hard to mitigate (unknown-knowns and unknown-unknowns) to the quadrants where they are easiest to mitigate (known-unknowns) or manage (known-knowns).

Systems Engineering Approach

What is Systems Engineering?

There are many different definitions of Systems Engineering, all of which provide different insights into the nature of the approach. Systems Engineering is needed in order to control undesirable effects (late emergence leading to costly rework) and manage complexity. It handles complexity by viewing the situation as a system and applying systems thinking in a structured way. It does go against the natural tendency to focus immediately on the complexities of the detailed technical solution. There is a need for concurrency and the use of judgment to take rational risks and make progress – but plans need to be made to reduce uncertainty in the understanding of the problem, or to reduce the consequences of this uncertainty – hence the need for technical risk management as part of the Systems Engineering toolkit.

Requirements are often unknown and vague at the start. Bone (7) proposes a model which can monitor and remove vagueness in requirements. The "Solutions" section of this paper describes some explicit techniques to handle and mature requirements. Even if the vagueness could be removed before flowing down the system solution to next stage and level of detail (and expense) it is vital to recognize that requirements will change. This is an aspect of a “wicked problem” as described by Conklin (8). Hence the issue of requirements changing or emerging has to be recognized and managed as a technical risk.

Program Management/Systems Engineering Interaction

Systems Engineering and Program Management should be thoroughly intertwined, and should be considered as two sides of the same coin. Program Management can become too driven by demonstrable progress, focusing too much on schedule and cost. Often, the need for and the role of the Program Manager is clearly understood, and that of the role of Systems Engineer is not.

The program plan must address all stakeholder needs and verification activities, and must mitigate risks. Risk consequences should be stated in terms of the requirements that will not be met.

In practice, this means moving forward with some level of risk and uncertainty. Therefore the plan must recognize the need to maintain / update the requirements (and add the lower levels of sub-system / component detail as they emerge), and to develop the missing parts of the requirements.

The fundamental message is that program plans and risk plans must be derived from a clear understanding of the requirements both technical and programmatic. Where technical risk or requirement uncertainty can have an unacceptable impact on the program outcome, then the plan must include activity to better understand the requirement or mitigate the risks.

Solutions

Technical Risk Maturity Assessment

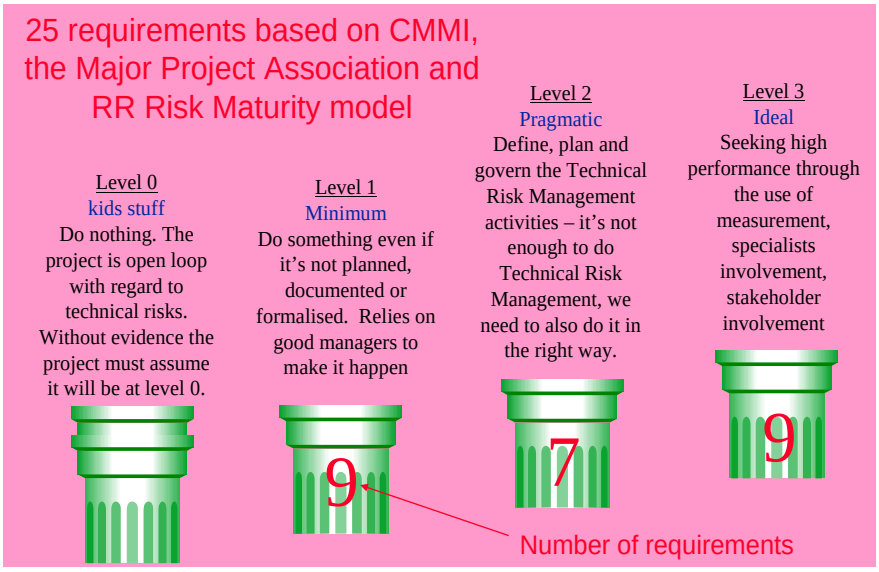

Figure 7. Technical Risk Maturity Assessment

Technical risk management is an essential tool for controlling uncertainty, scrap and rework. To accept this, an organization may have to go through a "cultural change", to reach a point where declaration and mitigation of technical risk is encouraged and supported.

One approach is to produce a "Technical Risk Maturity Assessment" to allow Projects to self-assess their capability at the start of the

project.

Risk Checklists

If the team has access to lists of typical risks and mitigations, organized hierarchically into classes, a more comprehensive identification of risks is achieved. The top level classes that we have identified are relatively domain-independent, and are shown in figure 8.

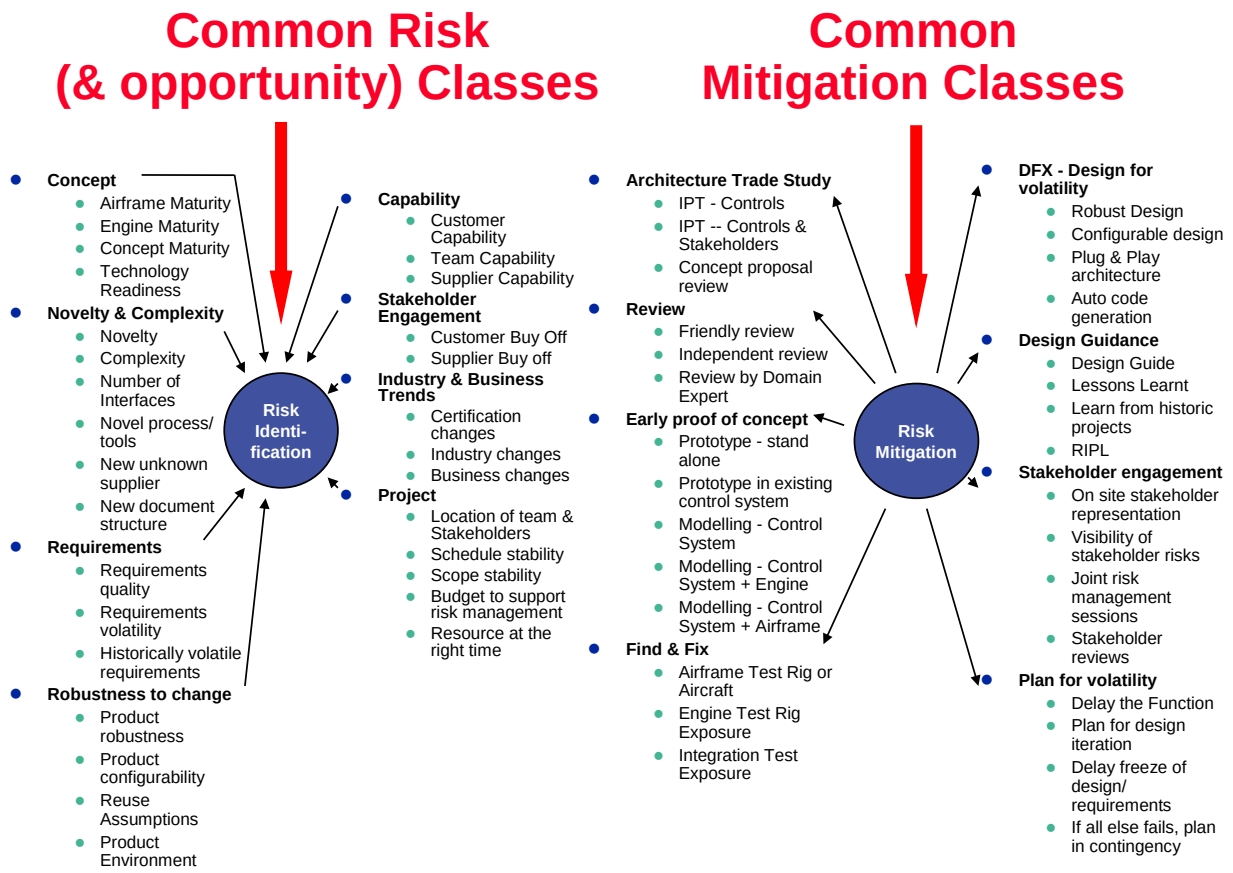

Figure 8. Risk and Mitigation Classes

The next level of the hierarchy is populated by reviewing the actions from technical gate reviews of previous system development programs. These risks are associated with each of the risk classes to make the first two levels of the hierarchy.

The final level in the hierarchy is populated by examining Lessons Learned and re-formatting each of these into a risk (i.e. the underlying problem) and an associated mitigation (the lesson). Figure 9 shows how these "mistakes" of previous projects can be fed back to help future projects manage risks.

The process recommended to Projects is to perform a risk brainstorm first, and then use the risk tool to "fill in" for any risks that were missed. This has the advantage that if the scope of the new project is broader than previous experience some thought is given to the specific risks associated with new functionality, and the risk tool is not applied "blindly".

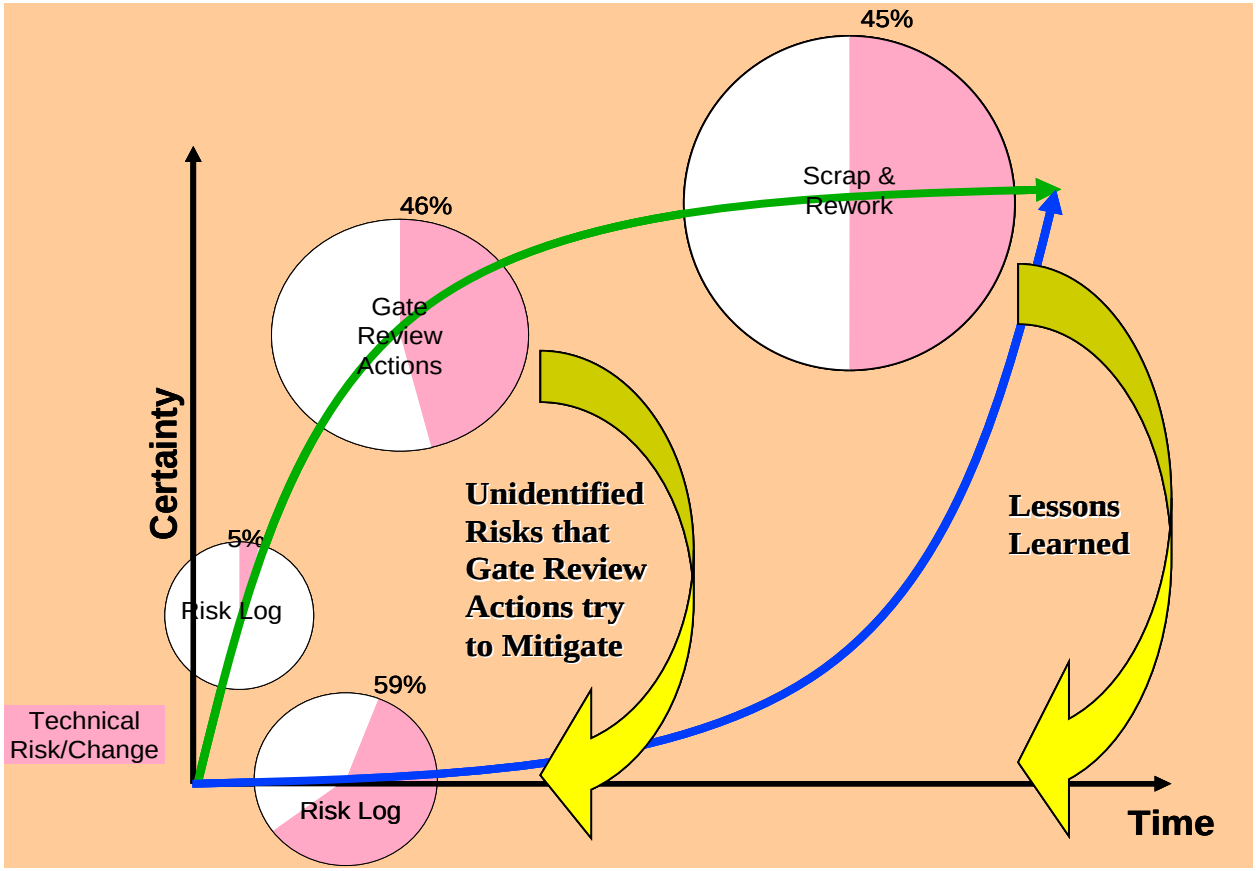

Figure 9. Risk Identification Sources

Metrics

Three measures are proposed to Projects to assess how well they are performing technical risk assessment:

- At the start of the Project, perform a Technical Risk Maturity Assessment and achieve an adequate level.
- Review the actions from the Technical Review Gates and count how many technical specific risks that were identified are not in the risk log.
- At the completion of the Project measure the scrap/rework rate

Does it Work?

Figure 10 shows a comparison of several incremental software build projects. All of the builds are of similar size, and were performed at a similar point in the project lifecycle, by the same software development team.

Two of the builds did not incorporate Technical Risk Management and risk mitigation practices, and show results typical of other software builds of this type - for the number of changes

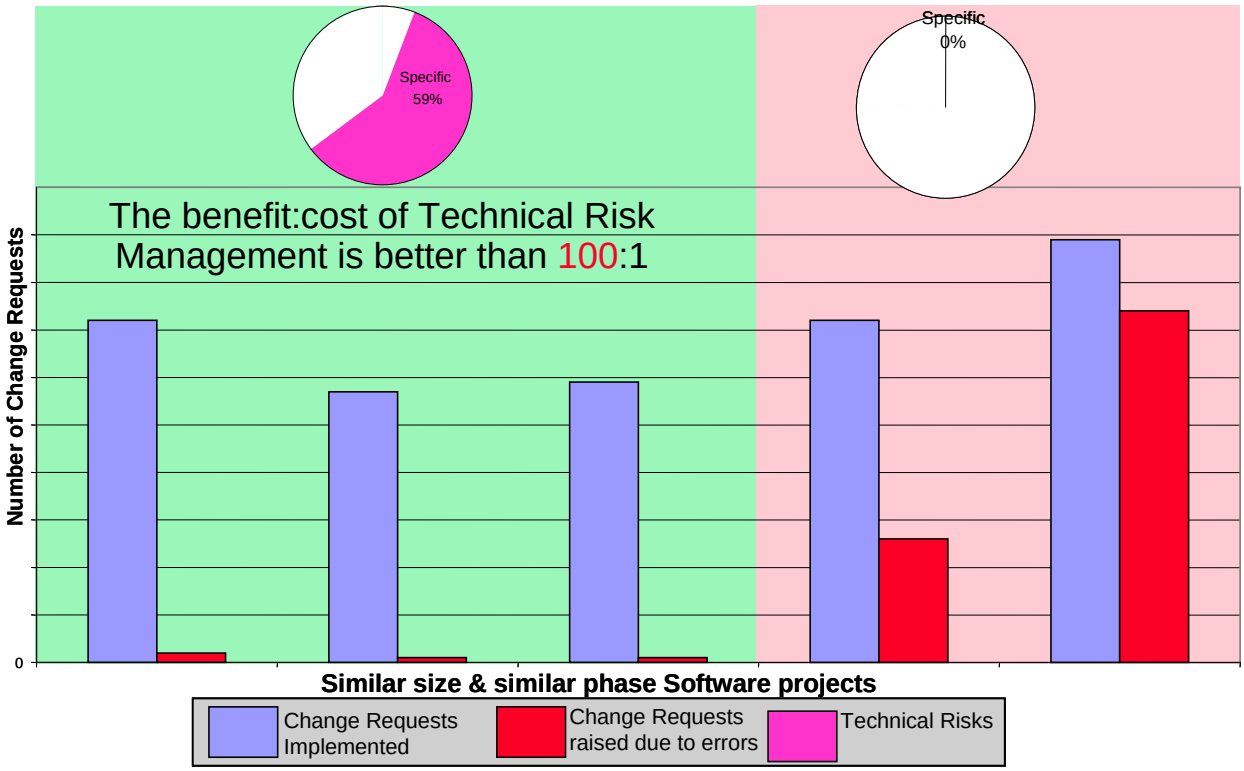

Figure 10. Technical Risk Management Works

incorporated in the build, around 30% to 70% additional changes were generated as a result of inadequacies in the implementation of these changes. Three of the builds made full use of Technical Risk Management and risk mitigation, and showed around 2% additional changes generated. The cost to perform the risk mitigation for each build was relatively small, sufficient to give benefit to cost ratios of greater than 100 to 1 - an investment that is hard to challenge!

Summary

- If no effort is made to control scrap and rework on a project, scrap rates of 50% can typically occur, leading to the program costing twice as much as it could have and requiring significantly more time to achieve a mature product.
- Contrary to expectations, changes in customer requirements are not a major driver of scrap and rework - most is internally generated by the development team.
- Most problems are detected during requirements or design review, but most escapes also occur in the review process. There appears to be no difference between software and hardware problems in this respect.
- Late detection of problems more than doubles the cost of correction.
- Systems Engineering and Technical Risk Management are critical in understanding and controlling the sources of scrap and rework

- Past experience (Lessons Learned, Technical Review Gate Actions) can provide a useful feedback mechanism to understand the technical risks that a new project may be facing
- Metrics are available to assess Technical Risk Management capability and effectiveness on a project
- Scrap and rework rates of less than 10% can be achieved, with benefit to cost ratios of better than 100:1

References

1. Buede, D and Larsen, R; "An Application of the CEaVa Method", INCOSE 10th International Symposium, Minneapolis, 2000
2. Buede, D and Larsen, R; "An Second Application of the CEaVa Method", INCOSE 11th International Symposium, Melbourne, 2001
3. McConnell, S; "Code Complete", Microsoft Press, ISBN 0 7356 1967 0, 2004. See Table 3.1 "Average cost of fixing defects based on when they're introduced and detected" - Adapted from "Design and Code Inspections to Reduce Errors in Program development (Fagan, 1976), "Software Defect Removal" (Dunn 1984), "Software Process Improvement at Hughes Aircraft" (Humphrey, Snyder and Willis 1991), "Calculating the Return on Investment from More Effective Requirements Management" (Leffingwell 1997), "Hughes Aircraft's Widespread Deployment of a Continuously Improving Software Process (Willis et al 1998), "An Economic Release Decision Model: Insights into Software Project Management (Grady 1999), "What We Have Learned About Fighting Defects" (Schull et al, 2002), and "Balancing Agility and Discipline: A Guide for the Perplexed (Boehm and Turner 2004)
4. Mawby, D and Stupples, D; "Deliver Complex Projects Successfully by Managing Uncertainty", Proceedings of EUSEC 2000, ISBN 3-89675-935-3
5. Schoening, W; "How Planning for Success Can Lead to Catastrophic Failure", INCOSE 16th International Symposium, Orlando, 2006
6. DeMarco, T and Lister, T; "Waltzing with Bears; Managing Risks on Software Projects", ISBN 0-932633-60-9
7. Bone, M; "Cyclone Process – Dealing with Vague Requirements", INCOSE 18th International Symposium, Utrecht, 2008
8. Conklin, J; "Wicked Problems and Social Complexity", from "Dialogue Mapping: Building Shared Understanding of Wicked Problems", Wiley, 2006. Also see the Cognexus Institute website, http://www.cognexus.org