When two is good company, but more is not a crowd

Andy J. Nolan*, Andrew C Pickard*, Jennifer L Russell[+] and William D Schindel[#]

*Rolls-Royce, [+]Parsons Brinckerhoff, [#]ICTT System Sciences

Andy.Nolan@rolls-royce.com, Andrew.C.Pickard@rolls-royce.com, RussellJe@PBWorld.com, Schindel@ictt.com

Abstract: This paper summarizes an approach to improve the effectiveness of the review (inspection) process. Effectiveness here is defined as the ability to reduce the number of defects escaping a review activity.

By carefully pairing up developers and reviews, Rolls-Royce was able to halve the rate of occurrence of defects in software, with no change to the process or tools, and with no changes to the team or the effort required to perform the reviews

The method hinges on an understanding of the capability of the developers and reviewers and making sure that only select pairings of team members will be allowed. The paper illustrates an example of the practice when applied to software code review but the principle can be applied to any development process. The paper ends by illustrating other ways to benefit from this approach.

Introduction

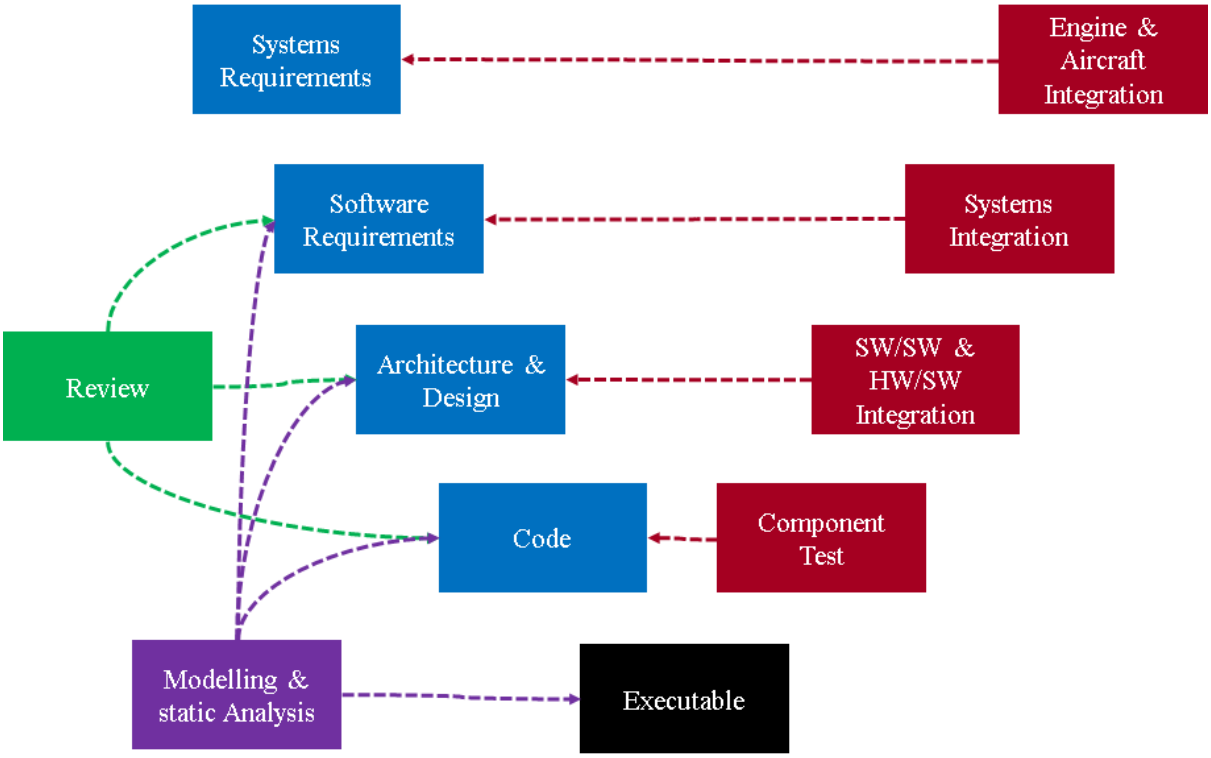

Figure 1: an overview of the software development process

At Rolls-Royce, we develop a range of software applications for embedded engine controllers. The applications are safety critical and satisfy Radio Technical Commission for Aeronautics' RTCA DO-178B level-A requirements (1) for Software Considerations in Airborne Systems and Equipment. The Technical Standard Order specifies the RTCA DO-178B for certification. The certification required places a heavy emphasis on gathering objective evidence and independent verification and validation.

While we make extensive use of testing for certification, we also rely on static analysis and independent reviews (see Figure 1). The net effect is that we have a very low defect escape rate of 0.03 defects per 1000 lines of code. This compares favourably with published industry values. World class organisations can produce software with a defect escape rate of 1 defect per 1000 lines of code. Rolls Royce's escape rate is even better than the well documented case of NASA (2 - 5), which delivers software with an average 0.1 defects per 1000 lines of code.

Verification is satisfied through review (all artefacts are placed under formal independent review), analysis (modelling, architectural analysis, static code analysis and Object Code verification) and test (Component test, software-software integration and hardware-software integration). Software-software integration is performed closed loop, with an engine model, in a host environment. Hardware-software testing is performed closed loop, with an engine model, in target on a real production quality Engine Controller.

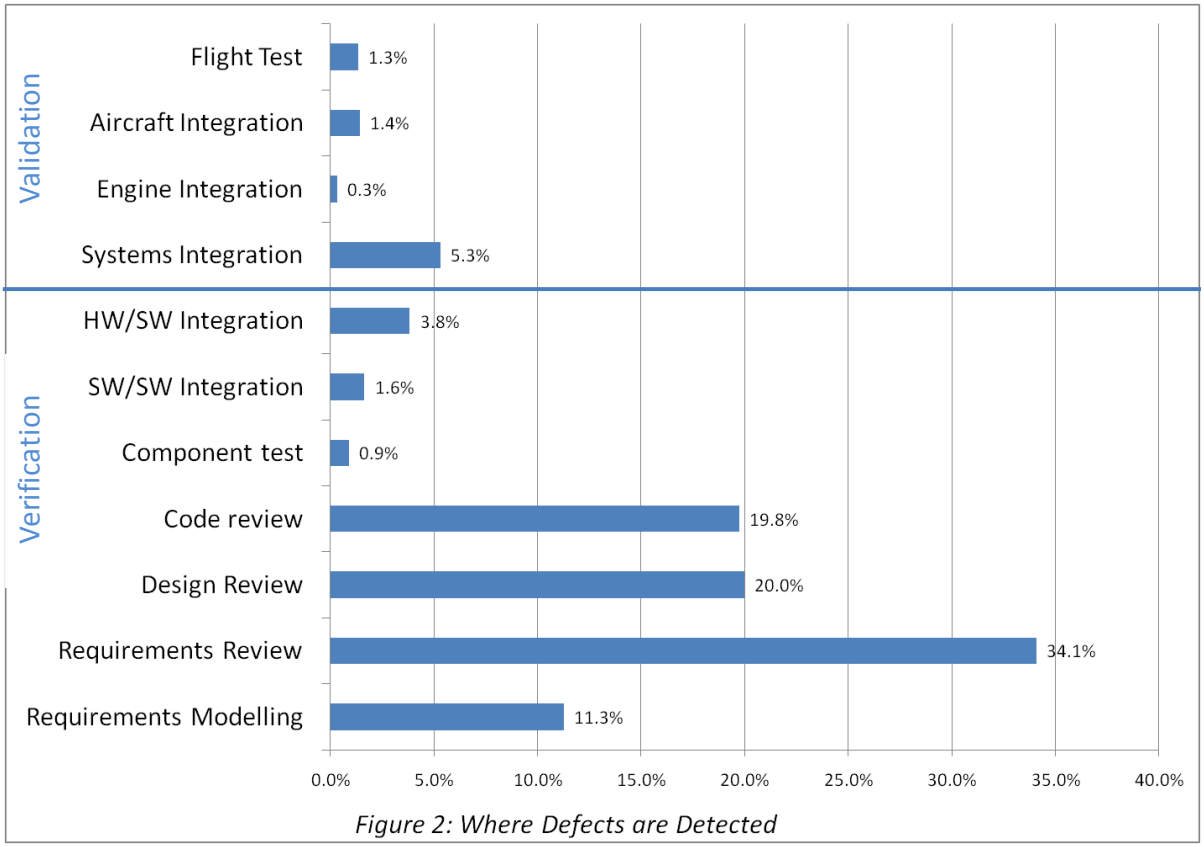

Figure 2: Where Defects are Detected

The software is then passed to the validation teams. The Systems Integration test is a closed loop environment, running on a production quality Engine Controller and has the capability of multiple fault insertion. Engine Integration is an environment where the production quality software and hardware are tested on a real engine. The Aircraft Integration combines the engine (or engine controller) with their aircraft. Flight test (not shown on Figure 1) is where the flight trials are performed.

Figure 2 shows, on average, where defects are detected during verification and validation. Within the software development process alone, we spend 52% of effort on testing and up to 24% on peer reviews.

Although we have an extremely low in-service defect rate, we do have internal defects, which are introduced, detected, and removed before the software team delivers the code to the system team.

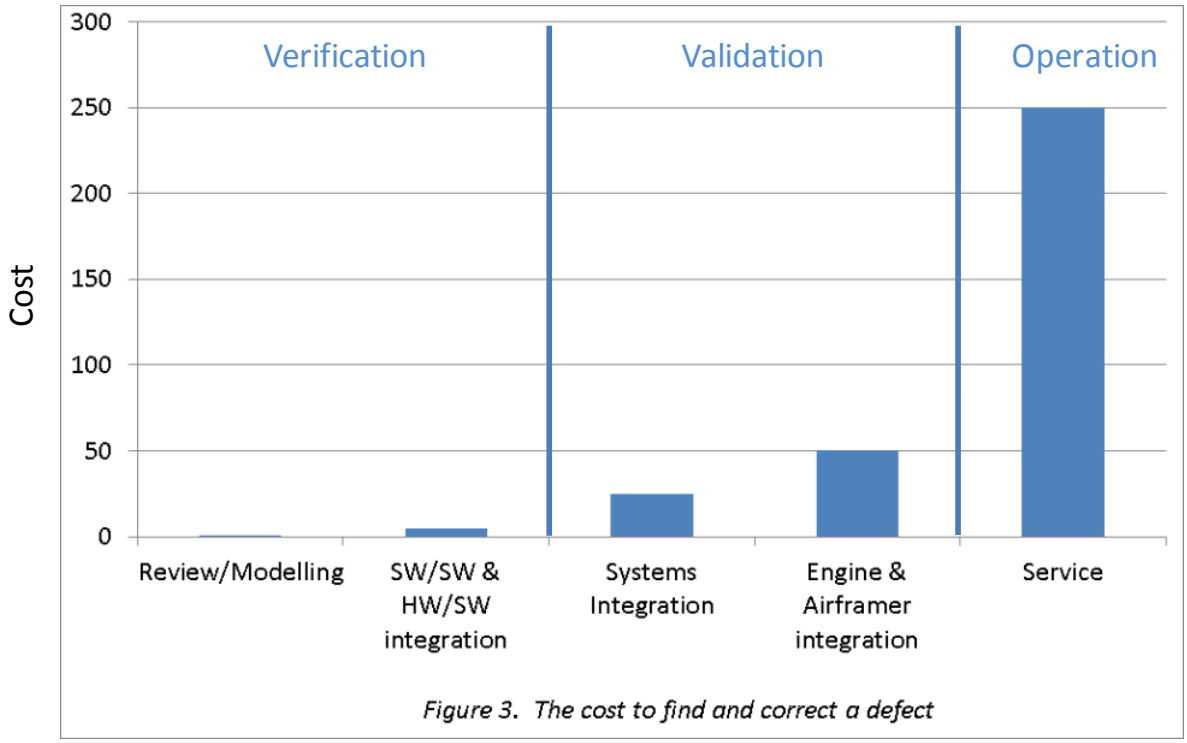

Figure 3. The cost to find and correct a defect

Figure 3 shows the cost to detect and remove a defect at various project stages. It is most cost effective for Rolls Royce to remove defects in the verification phase. Therefore, we seek to optimize defect detection early in the development lifecycle.

Throughout this paper, we have used the term defect to describe an unforeseen change to the software. However, on average 16% of the changes are caused by emergent needs that arise during engine and aircraft integration. In addition, 11% of software changes are to address hardware issues or improvements in system performance. For simplicity, this paper does not differentiate such changes, as all unexpected changes are costly.

This paper refers only to functional changes. A functional change is any change that will affect the object code (the compiled executable software). There are occurrences of defects with the documentation that do not result in a change to the software. These have been ignored primarily because functional changes tend to be more expensive to rectify.

By combining Figure 2 (where defects are detected) and Figure 3 (cost to detect and remove a defect) it is clear that a good review process is not only effective at removing defects, it is also cost effective (6).

Where found \ Were should have been found	Requirements Modelling	Requirements Review	Design Review	Code Review	Component Test	SW/SW Integration	HW/SW Integration	System Integration	Engine Integration	Flight Test
Requirements Modelling	185									
Requirements Review	129	458								
Design Review	36	128	77							
Code Review	8	78	32	102						
Component Test	0	1	3	9	4					
SW/SW Integration	1	45	61	60	0	30				
HW/SW Integration	1	10	55	30	0	0	13			
System Integration	18	145	18	5	0	19	1	41		
Engine Integration	10	11	14	0	0	3	3	3	4	
Flight Test	1	6	16	2	0	1	1	6	3	6

Figure 4: Where found / where should be found

Figure 4 was generated by adding two attributes to the change control system. Each defect raised recorded (1) where was the defect found and (2) where should the defect have been found. Figure 4 shows the two attributes ordered in time sequence where the top left are the first processes and the bottom right are the last. Clearly, defects on the diagonal were found in the correct place but any defect detected below this diagonal line was detected late.

Although the review process is one of the most effective V&V methods, when factoring for the cost of these late defect escapes (using data from Figure 3), it was evident that the review processes were about 50% effective by value.

The challenge to the team was to double the effectiveness of the review processes, specifically to halve the number of defect escapes.

Pairing Reviewers and Authors

Over the years, Rolls Royce has made many attempts to improve review effectiveness. The dominant approach was to refine the process and checklists used. Additional questions were incorporated in review checklists after analysing defect escapes. Over time, the checklists became long. Yet the review effectiveness did not improve. Therefore, this team considered an alternative approach to improve review effectiveness based on the team members' task performance and innate strengths.

Everyone is not equally capable of performing a specific task - people are good at different things. Anecdotally, we recognize that a person may be very good at contributing ideas and solutions at meeting, but terrible at following up with the details to execute those solutions. As managers and team mates, we may work around this, become frustrated, or just avoid inviting this team member to meetings. Regardless of how we respond, it is clear that this person is good at finding solutions, but we should not rely on this person to develop a detailed execution plan. However, not all strengths and weaknesses are equally visible through demonstration. Consequently, it is challenging to predict which tasks someone may be good at performing. Therefore, managers tend to rely on experience to predict future performance. If a team member has been successful in one area, often that person will continue to be successful in similar tasks. However, it is more difficult to predict if someone will be successful in different tasks.

Dozens of personality models provide personality and behaviour characterisation. Myers Briggs Type Indicator (MBTI) (7) is one such model. Myers Briggs describes an individual's behaviour preferences through 4 main categories and has been used throughout the business world for decades. Russell (8) offered a use of the MBTI characteristics by mapping the software development lifecycle to MBTI strengths. By understanding the personality characteristics and the tasks performed, it is possible to capitalize on behaviour preferences to maximize task performance. However obvious this may be, how many managers systematically take advantage of this fact when forming teams?

If we assume that an average review has an author and a reviewer, is it possible to combine these individuals to deterministically reduce the number of defect escapes? Is it possible that, by accident, a manager can put a weak author with a weak reviewer? Do managers actually know who is strong and weak? Rolls Royce conducted this study to determine if it was possible to quantitatively understand capability and use this information to form effective pairings of authors and reviewers to further reduce the defect escape rate.

The approach presented here to minimize defect escape rate from software code hinges on the need to define, understand, and manage the capabilities of the team members. The approach uses defect metrics to define the effectiveness of authors and reviewers. There are 3 steps involved:

Step 1 – Define team members

Quantify authors' effectiveness at creating defect free code

Quantify reviewers' effectiveness at detecting defects

Step 2 – Understand team members

Build defect models

Step 3 – Manage pairings

Apply author-reviewer pairing strengths

Consider behavioural preferences

Step 1 – Define team members

The first step in the process is to define a metric that can compare the effectiveness of the authors and reviewers.

Measuring Author Effectiveness

The author effectiveness measure relied on two factors, the "size" of the artefact and the number of defects found within it. Size was measured as lines-of-code modified or introduced – excluding comments and blank lines. The author effectiveness was measured as the number of defects introduced per 1000 lines of code. This measure did not account for complexity primarily because the relationship between defect rate and complexity is unclear. Previous studies have not shown that complex functions are more prone to defects. In this case, complexity was measured as cyclomatic complexity (a method commonly used for measuring the complexity of software source code).

The results from the measurement of author effectiveness showed that the difference in defect introduction rates between the best and worst varied by an order of magnitude i.e. the best author produced on average 0.5 defects per 1000 lines of code whilst the worst produced 18. Experience and training did not seem to be the dominant factor in determining the best authors. Actual author effectiveness rates are not shown in this research because the information is confidential to Rolls Royce.

Measuring reviewer effectiveness

The method for determining reviewer effectiveness was to find the average ratio of defects found in an artefact compared to the total number of defects found through other verification. For example, a reviewer may find five functional defects, but

through subsequent testing five more functional defects may be found. The reviewer would have found five out of the 10 defects resulting in a review-effectiveness of 50%.

The analysis was performed on historic work to prevent the research activity from affecting behaviour. When subsequent research was done with the team's awareness, we noticed additional improvements driven mainly by the challenge of beating the metric. This paper focuses on the initial study only.

Clearly, it will never be possible to determine exactly how many defects are in an artefact. Therefore, this study measured a "reasonable" time period over which we were confident we should have detected the majority of defects. Waiting a period of time minimised the risk of drawing incomplete conclusions.

The results yielded some surprises. A factor of three difference was measured in the defect detection rates of the reviewers; one reviewer could, on average, detect 90% of defects in an artefact whilst another detected only 36%. A second study of another software team showed a ten-fold difference in reviewer effectiveness. Remember that all reviewers are following the exact same process using the exact same checklists to guide them. Therefore, the difference in performance cannot be attributed to the processes used.

The best reviewers were not necessarily the most experienced or most trained. A new starter ranked in the top four, whist an experienced team member ranked in the bottom two. This demonstrates that experience and training is not necessarily the recipe for competence.

Eliminating process or training and experience as dominant factors for review effectiveness suggests that attitude or aptitude plays a dominant factor in competence. For example, the best reviewers all shared a common delight in finding defects and had a very detailed, critical approach to reviewing. Actual reviewer effectiveness rates are not shown in this research because the information is confidential to Rolls Royce.

Step 2 – Understand team members

To understand the team members, the effectiveness rates were organized from most to least effective. There were two people who were both authors and reviewers. The comparison of two subjects who were both reviewers and authors provides more insight into the concept that attitude or aptitude is a dominant factor in competence. Our data

	Author Effectiveness rank	Reviewer Effectiveness rank
Person 1	1	2 (*tie rating)
Person 2	2	
Person 3	3	
Person 4	4	
Person 5	5	2 (*tie rating)
Person 6	6	
Person 7		1
Person 8		4
Person 9		5
Person 10		6

Figure 5: Author and Reviewer Effectiveness ranking

revealed that competent authors are not necessarily competent reviewers. Of the six authors and six reviewers evaluated, the best author (1 of 6) was the second best reviewer (2 of 6). Two subjects had the same reviewer effectiveness rate. Another tied for second best reviewer (2 of 6) was the second worst author (5 of 6). Figure 5 shows the effectiveness ratings comparison for authors and reviewers.

It is clear that a competent author is not necessarily a competent reviewer. This supports the hypothesis that attitude or aptitude is an important factor in competence. An author tends to delight in creating something and the reviewers delight in finding errors in the creation. These attitudes are not always compatible. This is perhaps a warning to team leaders and managers who may assume that competence in one area assumes competence in all areas.

Build Defect Models

Based on the measures of author effectiveness and reviewer effectiveness, a matrix was formed for every pairing of authors and reviewer. The proprietary Rolls Royce data is undisclosed. Therefore, an example was recreated in Figure 6. In the example below, a team of 6 people, named A to F were assigned arbitrary effectiveness rates. The top left hand corner has been reserved for the most effective team members. In Figure 6, the vertical axis represents the number of defects introduced per 1000 lines of code by the author; the horizontal axis represents the effectiveness of the reviewer in finding those defects.

			Reviewer effectiveness defect detection rate					
			C	B	A	E	D	F
			94%	80%	75%	50%	45%	30%
Author Effectiveness Defects introduced per 1000 lines	A	0.5						
	B	1.0						
	C	3.0						
	D	4.0						
	E	10.0						
	F	18.0						

Figure 6: Author / Reviewer Matrix

			Reviewer effectiveness defect detection rate					
			C	B	A	E	D	F
			94%	80%	75%	50%	45%	30%
Author Effectiveness Defects introduced per 1000 lines	A	0.5	0.0	0.1	0.1	0.3	0.3	0.4
	B	1.0	0.1	0.2	0.3	0.5	0.6	0.7
	C	3.0	0.2	0.6	0.8	1.5	1.7	2.1
	D	4.0	0.2	0.8	1.0	2.0	2.2	2.8
	E	10.0	0.6	2.0	2.5	5.0	5.5	7.0
	F	18.0	1.1	3.6	4.5	9.0	9.9	12.6

Figure 7: Defect escape rates

It is now possible to hypothesise the number of defects that will escape from each combination of author and reviewer. For example, an author will produce "X" defects per 1000 lines of code. The reviewer has an effectiveness of "Y%" at finding those defects. Thus, for any author-reviewer pairing, the number of defects detected will be on average X*Y%. The number of defects escaping the review process will therefore be the number introduced minus the number detected = X – X*Y%.

Figure 7 then becomes a guide to team leaders and managers for how best to pair the authors and reviewers. Within Rolls Royce, we endeavour to always mix "green zone" combinations. The amber-zone is used in the event that pressures do not permit green-zone combinations. For a red-zone combination, we would pick a second reviewer (a third person) who was in the green-zone for that particular author, by reading across the row.

New-starters are still given the opportunity to develop and grow with no risk to the project or product quality. Where we had no readings for a new starter to a group, it was assumed by default that they belonged to the red-zone and therefore a second green-zone review was automatically performed.

This method does not require a change to the processes used to perform the reviewers and on average, the review process costs no more than normal as the reviews would have had to take place anyway.

Using the method above, it is possible to produce code with 1 defect per 1000 lines of code before it even leaves the coding team. The cost implications are evident from Figure 2 as we increase early defect detection and reduce defect escapes to more expensive phases

Validation of the matrix

Several approaches were used to validate the findings. Based on every author-reviewer pairing, it was possible to look at historical reviews and predict the number of defects that should have been detected based on the measures, and then to compare this to what was actually found. There was an R^2 correlation of 0.85, which would suggest that the model is reasonably good at estimating escape rates.

A further form of validation was to consider reviews just completed for a current build and, based on the size of the artefacts, determine the theoretical number of defects introduced based on the authors effectiveness. This figure was then compared to the actual number of defects found. We investigated any instances where there were two or more undetected defects per 1000 lines. Not surprisingly, each instance studied revealed an author-reviewer pairing that belonged to the red zone.

Identifying training needs

Although it was stated earlier that training was not necessarily a guarantee of a good author or reviewer, it is proposed that the "right" training is valuable. The measures

described here allow you to objectively measure the training provided to determine the “right” training.

In addition to using the matrix to manage the teams, the matrix is also used to target training needs. As training is only the beginning of someone’s development, not the end, we ensure that all defects found are fed back to the author for experiential learning. In addition, when we have an initial red-zone combination supplemented by a second review from a green-zone reviewer, additional defects were shown to the original reviewer, allowing them to learn where they missed defects.

Step 3 – Manage Pairings

A manager can use the matrix to identify pairing that will result in the highest likely defect detection rate. However, quantifying performance in this way is not always possible. And additional factors, such as software complexity and personal attitude may also affect the defect escape rate. Therefore, managers should consider many factors when assigning author-reviewer pairings.

Considerations

In many cases, there are multiple authors and/or reviews of a single software code package. It is still possible to derive effectiveness using statistical methods such as Design of Experiments. However, to be pragmatic, it is best to try to find examples with a single author and reviewer to calibrate the model.

Accounting for the function’s complexity

If people are not equal then the same can be said of software functions. Within Rolls-Royce**,** we performed an analysis of functions by understanding two factors; (a) the % of change and (b) the size of the function in terms of lines of code. We derived diagrams similar to that shown in Figure 8. The analysis revealed functional classes, e.g. functions with a similar characteristic, that had a similar volatility. With this information, we can anticipate which functions are most difficult to get right.

Understanding this provides a third variable in our defect reduction model i.e. never mix a difficult function with a weak author and reviewer. There must always be one point of strength**.**

The approach may not be to everyone’s taste. In some countries it will be illegal to capture performance measures. In most cases, regardless of its uncomfortable implications, many companies will just not have the data. Although the authors do not have evidence of the success of this strategy, it is perhaps still possible to derive a similar metric from qualitative data or simply from a manager’s observations. The principle of not have two weak people together still applies regardless of your philosophies, legislation or data.

Function size: Lines of Code	% change									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1000	100	200	300	400	500	600	700	800	900	1000
900	90	180	270	360	450	540	630	720	810	900
800	80	160	240	320	400	480	560	640	720	800
700	70	140	210	280	350	420	490	560	630	700
600	60	120	180	240	300	360	420	480	540	600
500	50	100	150	200	250	300	350	400	450	500
400	40	80	120	160	200	240	280	320	360	400
300	30	60	90	120	150	180	210	240	270	300
200	20	40	60	80	100	120	140	160	180	200
100	10	20	30	40	50	60	70	80	90	100

Figure 9: Function size vs Function change

Accounting for the human aspect

The work by Jennifer L. Russell (8) shows how a manager can use Myers-Briggs to understand the characters required for the various roles in a project. It is proposed here that the personality types for authors and reviews are different and a manager may be able to use Myers-Briggs to quickly understand how best to combine team members.

A post-processing effort to characterise the MBTI personality characteristics of each author and review led to the characteristics estimated in Figure 10.

		MBTI Type (estimated)
Authors	Person 1	ISTJ
	Person 2	ENxP
	Person 3	xNTP
	Person 4	xxFP
	Person 5	ISTJ
	Person 6	xNxP
Reviewers	Person 7	ISTJ
	Person 1	ISTJ
	Person 5	ISTJ
	Person 8	ISFJ
	Person 9	ISFJ
	Person 10	ENTP

x = not enough information to estimate this characteristic

Figure 10: Author and Reviewer MBTI estimation

An examination of the personality types reveals that there was a strong correlation between authors and the Perceiving (P) characteristic and Reviewers and the Judging (J) characteristic. The judging and perceiving characteristics describe how a person views the world around them. The Judging characteristic describes a person who considers the world to be fixed, hierarchical, and consistent. They pay attention to details, quickly identify inconsistency, and are bothered when they do not have order. The Perceiving characteristic describes a person who sees the world as flexible, and welcomes

change and variation. They are sensitive to the environment around them and can adapt quickly using their existing skills and knowledge.

The correlation between authors and the Perceiving characteristic is logical. Authors must be creative and adjust their knowledge of programming to create software solutions for the situation. This closely maps to the Perceiving characteristic as described by the MBTI. Therefore, it is logical, and supported by the correlation that people with a Perceiving characteristic would be effective authors.

Likewise, the correlation between reviewers and the Judging characteristic is also logical. Reviewers must be consistent and quickly identify errors and faults in the code. They rely on their knowledge and rigorously review the details of the code searching for errors and inconsistency. This also maps to the Judging characteristic as described by MBTI.

The strong correlation between the Perceiving characteristic for authors and Judging characteristic for reviewers may provide additional insight for managers. In addition to considering the defect detect rates, managers could consider personality characteristics in order to capitalize on behavioural preferences of the team. Pairing a Judging reviewer with a Perceiving author is likely to result in creative coding solutions that are carefully reviewed.

Managers should be aware to not over-rely on personality characteristics as a sole basis for pairing. It is possible that a judging person can be a good author and a perceiving person can be a good reviewer. People learn skills and grow from experience which may override some natural inclinations. This is evidenced in this research where the top author had a Judging characteristic instead of Perceiving. But, the natural inclinations can provide insight into where team members may be more comfortable and therefore, more likely to be successful. This provides insight and supporting detail to inform a manager before making pairings

Review Patterns and Pattern Based Systems Engineering Models

The above discussion on selection of reviews is based on patterns that the authors have identified relating to review in particular and more generally to System Verification and Validation. Figure 11 illustrates the first of these patterns:

The other four patterns share the same Problem Statement, Forces or Tensions and Context. The five patterns are:

Pattern 1: Who Reviews What?

Solution 1: Whoever is most able to detect the errors in a system artefact should review the artefact before it is used to create further system artefacts

Pattern 2: Effective Reviews Address Error Escapes

Solution 2: Measure the capabilities of team members in terms of rate of error introduction and rate of error detection. Never pair a developer who is prone to

introducing errors with a reviewer who is poor at detecting errors.

Evocative Name	**Who Reviews What?**
Problem Statement	Errors are introduced during development of a system that could have been detected by review. However, the errors are not found during review but are detected later in the development of the system, where the cost to fix them is considerably higher.
Forces or Tensions	There are two factors which impact error escapes; the rate at which errors are introduced, and the rate at which they are detected. Both of these are strongly impacted by the capabilities and behaviors of the people involved. It is rarely possible to assemble a team to develop a system where each person is equally capable, which means that inevitably there will be some system developers who introduce more errors and some reviewers who miss more errors than desired. Also, schedule pressures may mean that some members of the team who are best able to detect errors may not be available at the best time to perform the review
Context	It is inevitable that errors will be introduced during the development of a system. These may occur at any stage during system development, from requirements elicitation through design, integration, verification, validation and system deployment. Although it may be possible to deploy a system with known functional issues (consider many commercial software applications!) most systems cannot be deployed with unresolved functional issues. Also, there is a known relationship between the stage in the development process at which the problem is detected and the cost to fix the problem; finding a problem after entry into service costs between 200 (software) and 1000 (hardware) times as much to resolve than if the problem is detected and resolved during requirements elicitation and the early stages of design. However, the primary means of detecting errors early in the system development - review - is often not as effective as it could be, reflected in the number of issues that are detected later in the development process but should have been detected during requirements or design review.
Related Patterns	Effective Reviews Address Error Escapes. Train Your Reviewers. Train Your Developers. Danger! Difficult Function.
Solution	**Whoever is most able to detect the errors in a system artefact should review the artefact before it is used to create further system artefacts**
Examples:	1. The system verification team should review the system requirements for testability before they are used to guide the creation of the system design
	2. The systems architects/modelers should review the system requirements for completeness before they are used to guide the creation of the system design
	3. The system design team should review the system verification test cases and procedures for effectiveness and for "predictable failures" based on misunderstanding of how the system functions, prior to the start of system verification testing

Figure 11. System Development and Review, Pattern 1

Pattern 3: Train Your Reviewers

Solution 3: Use a formal mechanism to feed back errors detected by other means to the reviewer in a constructive way that will help them to improve their review effectiveness.

Pattern 4: Train Your Developers

Solution 4: Use a formal mechanism to feed back the detected errors to the system developers in a constructive way that will help them to improve their system development capability.

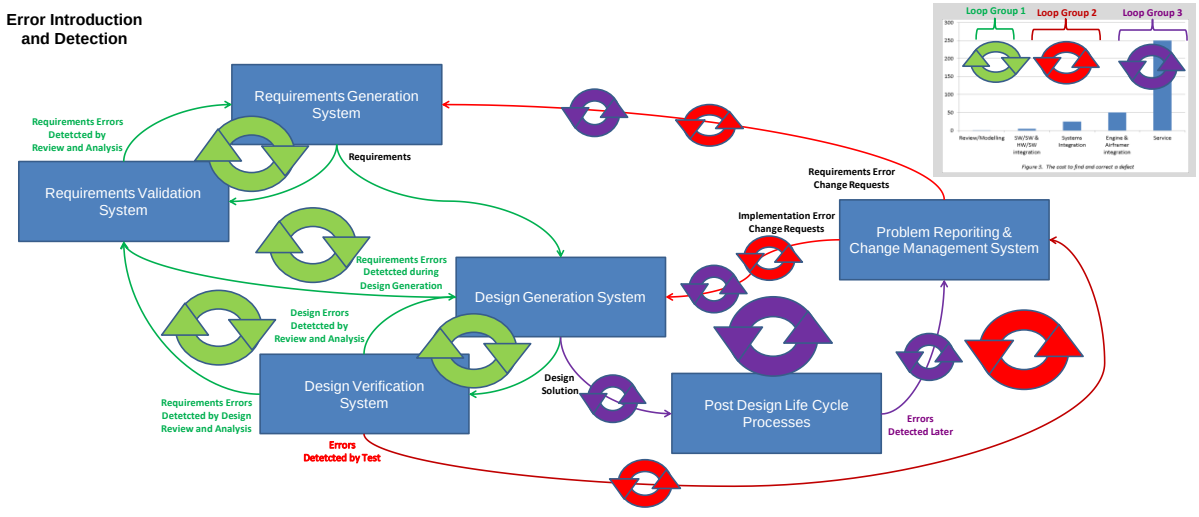

Figure 12. Verification and Validation Domain Model

Pattern 5: Danger! Difficult Function

Solution 5: When assigning work to be performed, do not assign development of a part of the system that is known to be difficult to get right (for instance, a difficult function in software) to the developer who is most likely to introduce errors or the

reviewer who is most likely to miss errors

Figure 12 shows a Pattern Based Systems Engineering (PBSE) domain model for Requirements Validation and Design Verification, including the primary pathways by which errors are introduced and detected. The aim of the five patterns described above is to try to contain error escapes to the green loops in figure 10, where the cost of rectification is lowest. The red loop corresponds to errors that escape the review and analysis process and are detected in system test, where they are correspondingly more expensive to rectify. The purple loop corresponds to errors that escape review analysis and test but are detected later, up to and including during service operation of the system. These are the most expensive to rectify, as illustrated in figure 3 (shown as a thumbnail in figure 10).

Figure 13 shows a role-based view of reviewer selection, capturing patterns 2 to 5 above. The “Single Review” path is abbreviated and does not show the feedback paths as this would over-complicate the diagram.

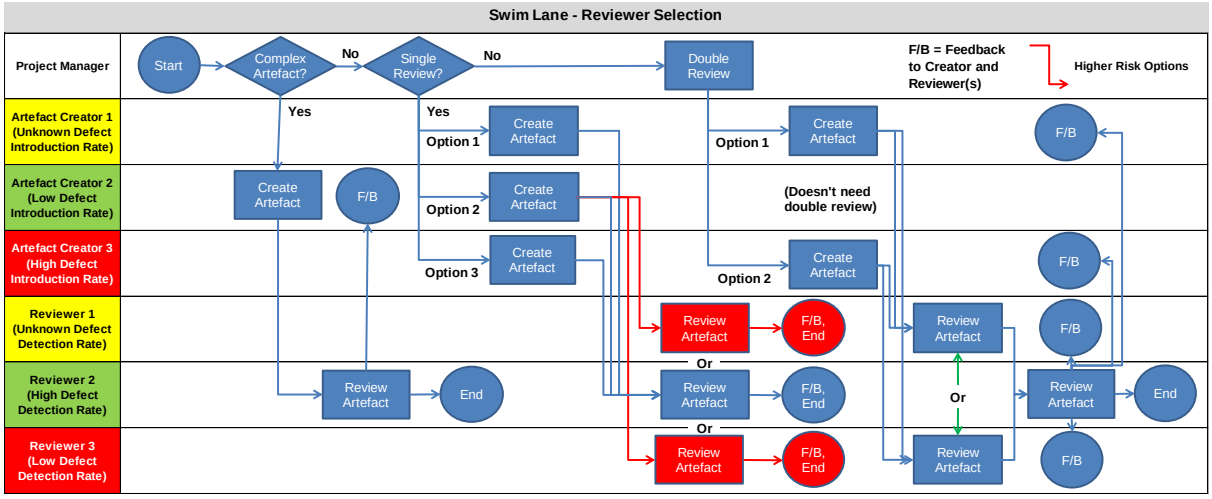

Figure 13. Role-Based Model of Reviewer Selection

Conclusions

Figure 14 shows the improvements achieved from applying the methodology to the requirements review and the design/code reviews. The baseline project did not use the approach. Projects 2, 3 and 4 applied the approach with refinements over time that increased overall effectiveness to 90%.

Previous papers published by Rolls-Royce (9, 10) have shown that the overall cost effectiveness of performing requirements review, uncertainty assessment and technical risk management is 100:1 return on investment. The improvement to defect detection illustrated in this figure is so significant that it is difficult to not justify the review efforts. The high cost of a defect escape and the review improvements justify adding more reviewers to an artefact.

If a project is struggling for time, cost and resource, then the method described in this paper will bring value with no additional impact to the project. However, if a project has headroom, then we recommend adding as many reviewers as you can.

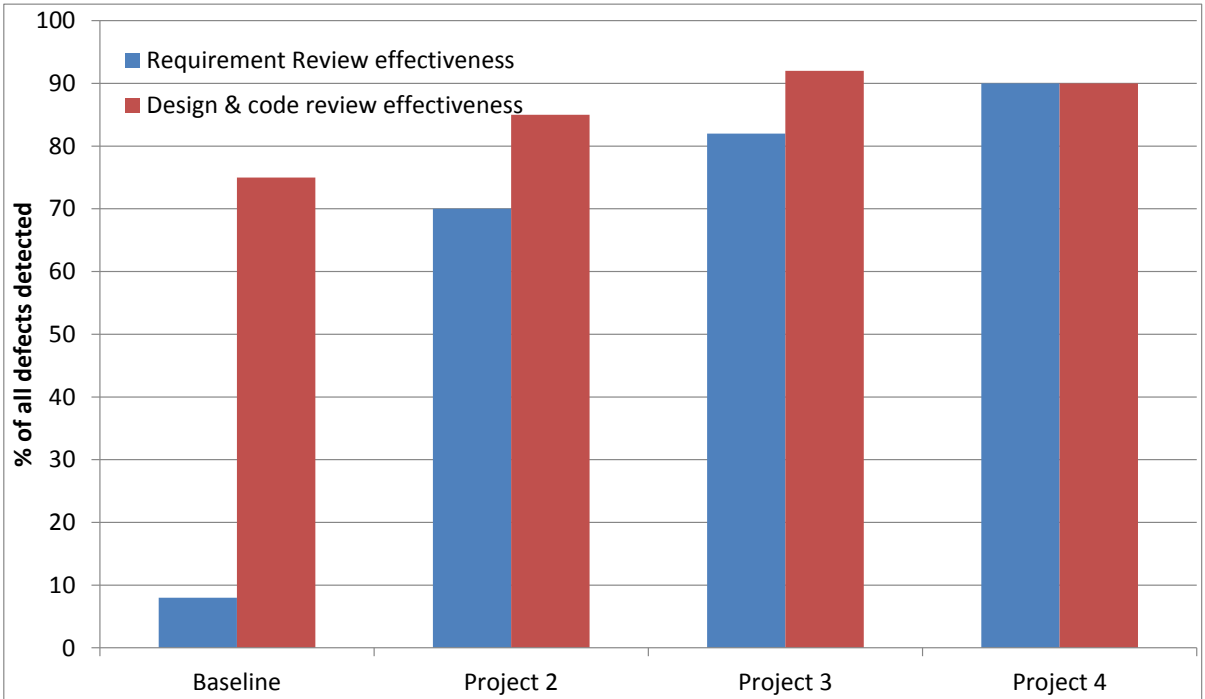

Figure 14. The effectiveness of the systems and software review processes. The values indicate the % of total defects detected by all V&V activities. The chart is in time sequence order from left

This paper has used the example of the code review process to illustrate the concepts. The principle is that there must always be at least one point of strength in any relationship. But there are other relationships to consider:

- The complexity of the task and the strength of the team developing it
- Mixing designers and testers
- Review of the system design by Systems Engineers, Systems Analysts and V&V team
- Involving Customer and Suppliers in reviews
- Personal strengths and preferences of the team

Defects do not occur by accident, but as a result of causes. Defects are predictable and controllable assuming you know the drivers that cause them. This paper concludes by demonstrating that competency is a key driver in defect escapes. It is in the interest and capability of a manager to manage the competency of their team.

The Pattern Based Systems Engineering model for Requirements Validation and Design Verification is in the process of being developed, as part of the INCOSE "Pattern Based Systems Engineering Challenge" (11) but is already showing some

powerful insights into the Verification and Validation process including detection of missing feedback loops and information flows.

References

1. RTCA DO-178B, “"Software Considerations in Airborne Systems and Equipment Certification”, RTCA Inc., December 1992

2. Madden, W.A. and Rone, K.Y. “Design, Development, Integration: Space Shuttle Primary Flight Software System,” Communications of the ACM 27, No. 9 (September 1984), p. 918

3. Joyce, E.J. “Is Error-Free Software Achievable?” Datamation, 15 February 1989, pp. 53–56

4. Kolkhorst, B.G. and Macina, A.J. “Developing Error-Free Software”, IEEE AES Magazine, November 1988, pp. 25–31

5. Macina, A.J. “Independent Verification and Validation Testing of the Space Shuttle Primary Flight Software System” Houston, Texas, IBM, 28 April 1980

6. Pickard, A.C. and Nolan, A.J. “How Cost Effective is your V&V?”, INCOSE 23rd International Symposium, Philadelphia, 2013

7. Myers, I.B. and Briggs, P.B. “Gifts Differing: Understanding Personality Type”, Mountain View, CA. Davies-Black Publishing. ISBN 0-89106-074-X, 1980

8. Russell, J.L. “What color is your nail polish? How to use Myers-Briggs personality characteristics to identify potential Systems Engineers in your organization”, 24th INCOSE International Symposium, Las Vegas, 2014

9. Pickard, A.C., Nolan, A.J. and Beasley, R. “Certainty, Risk and Gambling in the Development of Complex Systems”, 20th INCOSE International Symposium, Chicago, 2010

10. Nolan, A. J. and Pickard, A.C. “Reducing Scrap and Rework”, 23rd INCOSE International Symposium, Philadelphia, 2013

11. Web site of the INCOSE Patterns Challenge Team, with references: http://www.omgwiki.org/MBSE/doku.php?id=mbse:patterns:patterns

Biography

Andrew Nolan joined Rolls-Royce in 1989 after completing a degree at Sheffield University. He is the Chief of Software Improvement for Rolls-Royce based in the UK. He is a Fellow of the British Computer Society and a chartered Engineer in software engineering. Andrew has spent over a decade managing large scale software projects as well as a decade improving the way Rolls-Royce manages projects.

Andrew Pickard joined Rolls-Royce in 1977 after completing a Ph.D. at Cambridge University in Fatigue and Fracture of Metals and Alloys. He is a Rolls-Royce Associate Fellow in System Engineering, a Fellow of the Institute of Materials, Minerals and Mining, a Chartered Engineer and a member of SAE International and of INCOSE. He is Vice-Chair of the SAE Aerospace Council and represents Rolls-Royce on the INCOSE Corporate Advisory Board.

Jennifer Russell, EISE, has 20 years of experience in transportation, including as a Captain in the U.S. Army for 8 years. In her 9 years with Parsons Brinckerhoff, Jennifer focused on transit and rail planning and systems engineering. Jennifer holds a B.S. in Engineering Psychology from the United States Military Academy and a M.S. and Engineer Degree (2007) in Industrial and Systems Engineering from the University of Southern California.

William D. (Bill) Schindel is president of ICTT System Sciences. His engineering career began in mil/aero systems with IBM Federal Systems, included faculty service at Rose-Hulman Institute of Technology, and founding of three systems enterprises. Bill co-led a 2013 project on the science of Systems of Innovation in the INCOSE System Science Working Group. He co-leads the Patterns Challenge Team of the OMG/INCOSE MBSE Initiative.