LEVERAGING NATURAL LANGUAGE PROCESSING IN REQUIREMENTS ANALYSIS:

How to Eliminate Over Half of All Design Errors Before they Occur

INTRODUCTION

Numerous studies have shown that the cost of fixing engineering errors in systems and software increases exponentially over the project life cycle. Couple that with results showing that more than half of all engineering errors originate in the requirements, and you have a compelling argument in favour of finding and correcting requirements errors where they occur… at the very beginning of the project.

Up until recently, however, most error detection tools used in systems development – code syntax checkers, debuggers, static analysis tools, wiring testers and the like – have been designed to find errors in the software or hardware build rather than in the requirements.

Automated detection of requirements errors has been a much tougher nut to crack. Most requirements documents are still written in natural language, and often, it's the inherent ambiguities of natural language that cause requirements errors. Finding ways to analyze natural language text and identify possible sources of requirements errors has been a difficult problem to solve.

Fortunately, new requirements analysis tools based on natural language processing (NLP) are now emerging. They promise to significantly reduce the cost of fixing requirements errors by finding them earlier and faster, and to free domain experts from tedious, time-consuming tasks that waste their expertise.

COST TO FIX ERRORS RISES EXPONENTIALLY OVER THE DEVELOPMENT CYCLE

In 2004, NASA performed a study on the relative cost of fixing engineering errors during the various phases of a project development cycle[vii]. They reviewed a number of previous studies (Boehm[viii], Rothman[ix] , McGibbon[x], Chigital[xi], and others), and also ran cost analyses on a number of large systems development projects

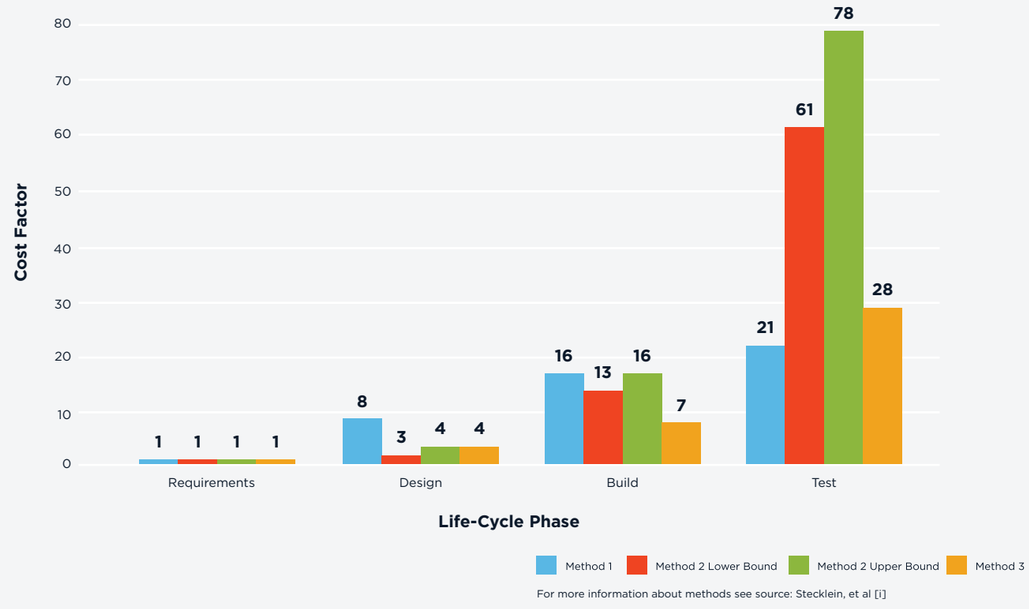
There was one finding common to all the software studies they examined and all the systems development projects they studied: the cost to fix software defects rose exponentially with each successive phase of the project life cycle. Figure 1 shows a comparison of the system cost-to-fix results (excluding operations) NASA obtained from their various methodologies, while Figure 2 compares their system results with the software cost models they found in the earlier studies they had examined.

Looking at these findings, it's easy to understand why companies would want to find and fix errors more efficiently in the later phases of the project life cycle – the build/code and test phases – where costs-to-repair rise astronomically, and before the product goes operational, where they rise even more. Thus we see an emphasis on automated tools like code syntax checkers, debuggers and test coverage tools in those phases.

But the fact is, most systems and software defects that are found in those phases – or in the operations phase – do not originate in those phases.

Figure 1: Comparison of System Cost Factors – Excluding Operations

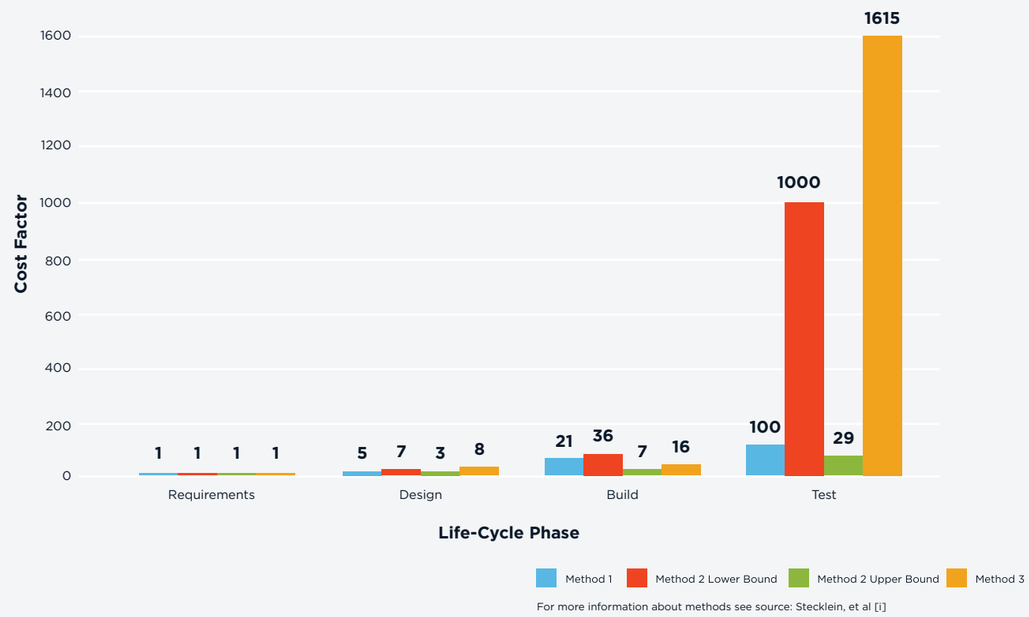
(Source: Stecklein, et al [i])



For more information about methods see source: Stecklein, et al [i]

Figure 2: Comparison of Software and System Cost Factors

(Source: Stecklein, et al [i])



For more information about methods see source: Stecklein, et al [i]

OVER HALF OF ALL ENGINEERING ERRORS ORIGINATE IN THE REQUIREMENTS

The vast majority of engineering defects detected toward the end of a project or in operations were actually present much earlier – in the requirements and design phases – and could have been corrected at far less expense had they been detected earlier.

A study by Pulitzer Prize-winning IT consultant and author James Martin[xii], for example, found that:

- The root cause of 56 percent of all defects identified in software projects are introduced during the requirements analysis and definition phase (Figure 3).
- About 50% of requirements defects are the result of poorly written, unclear, ambiguous or incorrect requirements.
- The other 50% are due to incompleteness of specification (incomplete and omitted requirements.
- 82% of application rework is related to requirements errors.

Just looking back at the NASA data discussed earlier, we can see that companies could gain significant savings by finding and correcting requirements errors near their point of origin, in the requirement analysis and definition phase of the project. Other studies suggest there are additional high premiums to pay for undetected requirements errors.

A study by IAG consulting, which analyzed "the importance and impact of business requirements on enterprise success with technology projects" found that 68% of companies suffer from poor requirements specifications practices, and that these companies:

- Spent 49% more money to deliver applications
- Took 39% more time to deliver applications
- Reported 79% of their projects over time and over budget
- Consumed over 41.5% of its new project development resources on poorly specified requirements

From the data they collected, IAG concluded that...

> *"There is a 60% time and cost premium to be paid on projects with poor quality requirements."*

It is readily apparent that companies need to do more than they have in the past to ensure that they are authoring and accepting high quality requirements.

Figure 3: Time and cost premiums on low quality requirements
(Source: IAG [iv])

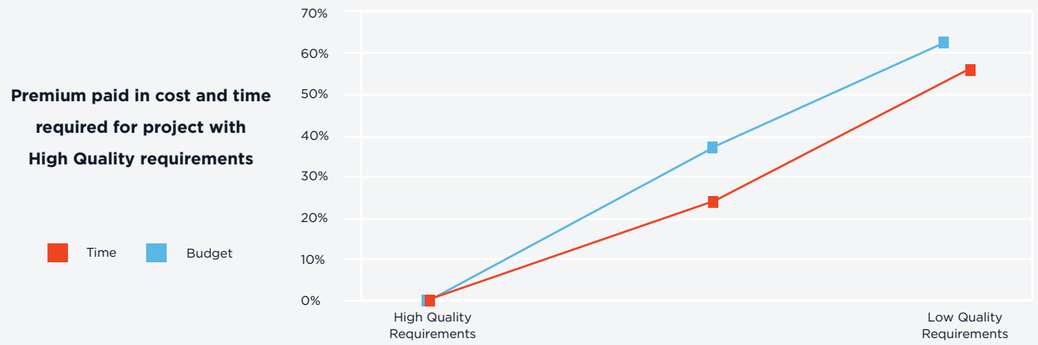
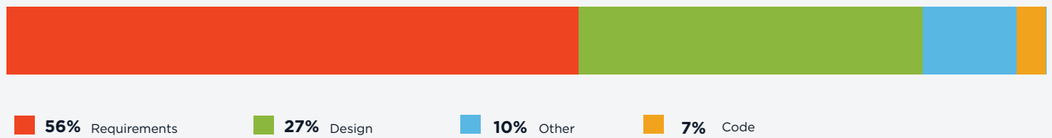


Figure 4: Distribution of defects in software projects by development phase
(Source: Martin [iii])



Model-based Specifications

One way companies have tried to combat the problem of requirements errors is through the use of formal specification methodologies like Model-Based Systems Engineering (MBSE). In MBSE, domain models – rather than natural language (NL) requirements documents – are used as the primary means of communication between engineers. Since the specification language is essentially mathematical and the domain models can be tested and verified, there is little room for ambiguity and far less chance that a requirement error will not be caught.

Even where such methods are used, however, the initial, top-level requirements are always stated in natural language. This is normally the case in defence, space and other industries where the original requirements come from outside the supplier organization. And in these situations, the supplier usually has a contractual obligation to trace the elements of the domain model to the natural language customer requirements they fulfil.

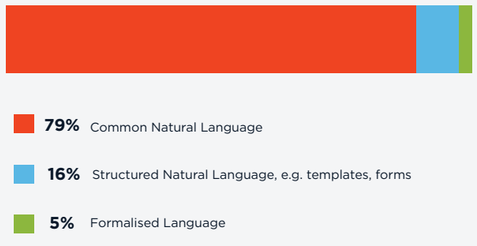
In such cases, vague or ambiguous requirements in the natural language specification will almost certainly slow the development of the domain models. They could also impede traceability between the domain models and the customer specification.

Worse yet, vagueness or ambiguity in the NL source requirements could introduce errors into the models through misinterpretation. And even when the NL source requirements are clear, it is extremely difficult to accurately translate the semantics of natural language into a mathematical model, and trying to do so tends to increase the size and complexity of the model dramatically. As professors Shilpi Singh and Lakshmi Saikia point out in a recent paper, "Formal methods help in writing specifications that are not always identical to the stated requirements." [xiv] This latter problem is only exacerbated by ambiguity in the NL requirements. Thus, even formal environments are not completely immune to errors caused by ambiguity in NL specifications.

Formalised specifications, however, occupy only a small portion of the specification universe. The vast majority of requirements documents are still written in natural language.

In fact, one recent study[xv] found that 79% of companies were using "common" (unstructured) natural language in their requirements documents, while 16% used "structured" (restricted) natural language, employing templates and forms. Only 5% of the companies surveyed said they were using formal approaches like MBSE (Figure 5).

Figure 5: Time and cost premiums on low quality requirements
(Source: Mich, et al [v])



With natural language dominating the requirements definition space, it's only natural that the dominant methods for finding errors in requirements specifications have been those aimed at analysis of NL requirements. Historically, those methods have been based on human review of the requirements. And most of them employ one or both of two techniques: (1) checklists and (2) peer review.

Checklists

Checklists are valuable, not only for analysis of an existing requirements document, but also for training new engineers in requirements engineering (RE) best practices. Many organizations concerned with the building of large, complex, requirements-driven systems publish some kind of requirements quality checklist. NASA, for example, includes three such checklists as an appendix (*How to Write a Good Requirement*) to their *Systems Engineering Handbook* [xvi]. INCOSE has published a 74-page *Guide to Writing Requirements* [xvii]. Even we at QRA Corp have published our own guide to help systems engineers write clear requirements. [xvii]

The problem with this method is that it is difficult and time consuming to review every requirement of a large specification manually, even with a checklist. This is due, in part, to the size of such checklists. Consider the following:

- INCOSE's Guide for Writing Requirements lists 44 "rules" to observe when writing a requirement.
- NASA's three checklists encompass 38 check points.
- QRA's guide includes 21 points.

Failure to keep so many rules in mind when writing requirements is normal. Manually checking every requirement against so many "rules" or "best practices" is time-consuming, tedious and an inefficient use of valuable domain expert resources.

Peer Review

Peer review is much like checklist review, but enhanced by "parallel processing" (multiple pairs of eyes) and a variety of perspectives.

Shilpi Singh and Lakshmi Saikiav propose that the most effective means of spotting ambiguities in requirements may be to:

> *"...hand them over to several stakeholders, ask each for an interpretation, and compare these interpretations afterward. If the interpretations differ, the requirements are ambiguous."*

The problem here is that peer review doesn't eliminate the problems of checklist review. In fact, in terms of expense, it multiplies them. As Singh and Saikia go on to point out, "this approach is economically feasible only for small sets of requirements." [v] In other words, peer review increases the chances of error detection, but it also multiplies labour cost.

Manual review of any kind can also be a fatiguing and extremely time-consuming (expensive) process when dealing with a large document of newly-defined customer requirements for a complex system.

A NEW, BETTER SOLUTION: NLP REQUIREMENTS ANALYSIS TOOLS

What's needed, we believe, is an automated way to help engineers and project managers author and clean up natural language requirements – make them crystal clear, thus easier to understand and evaluate – before putting them out for peer review, before modelling and design begins.

As widely-cited IT author Capers Jones points out:

"Far too much of the software literature concentrates on code defects and ignores the more numerous defects found in requirements and design. It is also interesting that many of the companies selling quality tools such as static analysis tools and test tools focus only on code defects.

> *Unless requirement & design defects are prevented or removed before coding starts, they will eventually find their way into the code where it may be difficult to remove them."*

Fortunately, a new class of tools is now emerging that addresses this problem. This new tool class makes use of natural language processing (NLP) techniques to help system engineers and project managers refine natural language requirements in much the same way that syntax checkers and debuggers help software engineers refine their code.

These new tools – called NLP requirements analysis (RA) tools – analyze the language used the specification of individual requirements. They then provide the user with a quality assessment of each requirement analyzed. These assessments flag any language usage (or lack thereof) within the requirement that may indicate a violation of requirements engineering (RE) best practices within the organization. In other words, they automate and significantly speed the task of searching for possible errors in NL requirements documents.

NLP RA tools offer **three major benefits** to systems engineers and project managers tasked with RE duties.

First, NLP RA tools **analyze requirements instantly.** Even very large requirements documents with thousands of requirements can be evaluated in seconds. Systems engineers and project managers get instant feedback on all the requirements they've authored or need to analyze.

Second, the reports these tools generate **show exactly where work is needed.** They provide visual scoring of each requirement assessed. Engineers can see immediately which sections of the document and which specific requirements need the most work.

Finally and most importantly, these tools **automate a tedious task** that doesn't require domain expertise. Manual review of requirements documents – even portions of those documents or changes to them – is a fatiguing and time-consuming task when one is armed only with a long checklist of RE best practices. It's a waste of a domain expert's valuable time and know-how.

That's not to say that human review of requirements is unneccessary or unimportant. Rather, these new NLP tools will free engineers from the "menial" portion of this task: sifting through every single requirement and making sure each is written to organizational guidelines and best practices. They will let domain experts focus on what's really needed: reviewing the results the tools provide and – taking their cues from the "red flags" generated by those tools – using their expertise to correct the deficiencies they find.

AN IN-DEPTH LOOK AT NLP REQUIREMENTS ANALYSIS

The idea of using NLP in requirements analysis isn't altogether new. NASA built and conducted studies with such a tool in the late 1990s.[xx] They called their software Automated Requirements Measurement, or ARM.

The NASA researchers, led by William Wilson, Linda Rosenberg and Lawrence Hyatt, clearly saw a need for such tools, in spite of the growing interest in formal specifications at that time:

"Despite the significant advantages attributed to the use of formal specification languages, their use has not become common practice. Because requirements that the acquirer expects the developer to contractually satisfy must be understood by both parties, specifications are most often written in natural language..."

> *"...The use of natural language to prescribe complex, dynamic systems has at least three severe problems: ambiguity, inaccuracy and inconsistency."*

NASA identified a series of "quality attributes" that requirements documents should possess. These desirable properties were:

- **Complete** - precisely defines the system's responses to all real-world situations the system will encounter.
- **Consistent** - does not contain conflicts between requirements statements.
- **Correct** - accurately identifies the conditions of all situations the system will encounter and precisely defines the system's response to them.
- **Modifiable** - as a logical structuring with related concerns grouped together.
- **Ranked** - organizes the specification statements by importance and/or stability (which may conflict with the document's modifiability).
- **Traceable** - identifies each requirement uniquely.
- **Unambiguous** - states all requirements in such a manner that each can only be interpreted one way.
- **Valid** - all project participants can understand, analyze, accept or approve it.
- **Verifiable** - must be consistent with related specifications at other (higher and lower) levels of abstraction

The NASA researchers realized, however, that most, if not all, of these characteristics are subjective, and therefore difficult to measure. So, they went on to identify a set of "quality indicators" that could be associated with the desired quality attributes.

These quality indicators are extremely useful in NLP requirements analysis, because they "can be found and counted with relative ease" and their presence, absence, abundance or dearth within a given requirement or the document as a whole tends to indicate the presence or absence of a related quality attribute. Wilson, Rosenberg and Hyatt grouped the quality indicators they found into nine categories. Four of these categories – size, readability, specification depth and text structure – reflect upon the structure of the specification document as a whole and do not apply to individual requirements. The other five quality indicator categories relate to the quality of individual requirements statements. It is this second group that is of great interest to requirements analysts and thus to these new NLP tools.

The requirements statement quality indicator categories are:

- **Imperatives** – words that give a command like *shall, must, will, etc.*
- **Continuances** – words like *below:, as follows:, following:, etc.,* which introduce the specification of requirements at a lower level, excessive use of which may indicate overly complex requirements.
- **Directives** – words or phrases like *figure, table and for example,* which point to illustrative information within the document and thus tend to indicate requirements that are more understandable.
- **Options** – words like *can, may and optionally* that appear to give the supplier latitude in satisfying the requirement and thus reduce the acquirer's control over the system.
- **Weak Phrases** – words and phrases like *adequate, as a minimum, easy, normal, etc.,* that can cause uncertainty and leave room for multiple interpretations.

The words and phrases in this second class of quality indicators, those in the categories related to individual requirements, appear in great abundance in requirements documents. This makes these quality indicators tedious to search for manually, but prime targets for NLP analysis.

The results of NASA's ARM tool study showed that the quality of requirements documents and of individual requirements statements can, to a certain extent, be quantified and evaluated using such quality indicators.

Unfortunately, no one was able to pick up the ball and run with it at that time.

While the ARM study has been frequently cited, and subsequent studies (Mich[v], Singh & Saikiaix[ix], Kof[xxi], Sateli[xxii]) have looked at the feasibility of NLP tools for improving requirements analysis and authoring, there was little further development of NLP requirements analysis tools for many years, as the computing power at the disposal of most requirements authors was insufficient for effective use of NLP algorithms...

Until now.

Today, however, computational advances are finally allowing modern NLP requirements analysis tools to emerge. These new applications are built on the same principles as NASA's ARM tool. They evaluate requirements based on the same or similar quality indicators. But now they are made for every-day requirements analysis and authoring.

These new NLP RA tools normally possess five key attributes:

- **Specific** – designed for daily use in requirements analysis, authoring and editing; optimized to find and evaluate the properties of quality requirements
- **Simple** – light-weight applications with highly intuitive user interface; easy to learn and use
- **Seamless** – work with existing requirements authoring tools as an add-on or extension, or through easy import/export
- **Fast** – provide immediate, on-demand analysis
- **Configurable** – Easily configured for domain-specific terminology and practices; adaptable to changing user needs and preferences

As an example of this emerging class of tools, we'll look at QRA Corp's new NLP requirements analysis product: QVscribe. QVscribe has been initially designed as an extension to Microsoft Word, the world's most popular requirements authoring platform. Plugins for other authoring and RM tools are currently under development.

QVscribe can be easily configured to a company's best practices through its configuration dialog box (Figure 6).

Figure 6: Configuring QVscribe to company practices

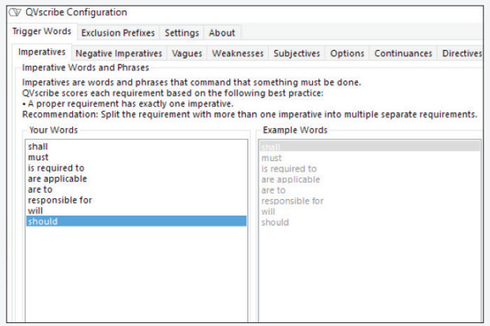
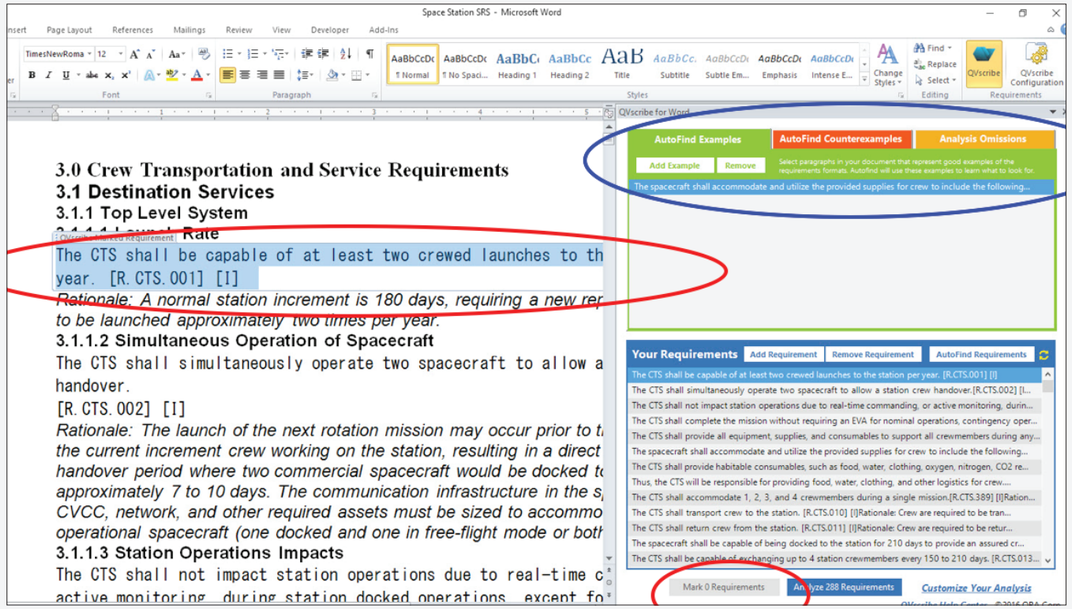


Figure 7: Autodetecting and marking requirements in QVscribe



ISS Crew Transportation and Services Requirements Document (Source: NASA)

The user or his organization supplies a few examples of how requirements are identified within the organization or project. Counterexamples can also be supplied to exclude certain constructions or sections from autodetection (Figure 7, blue ellipse). Requirements can also be marked manually by highlighting the requirement text and clicking on the Mark Requirements button (Figure 7, red ellipses).

Once the user has marked all the requirements he or she wishes to analyze, clicking on the Analyze Requirements button at the bottom of the QVscribe window initiates the analysis process. When QVscribe completes its analysis – usually within a few seconds – it presents the user with a score – from one to five bars – for each requirement analyzed (Figure 8).

Clicking on any requirement score in the analysis pane highlights the requirement and the quality indicators within it that triggered the given score (Figure 9).

Figure 8: QVscribe visual scorecard

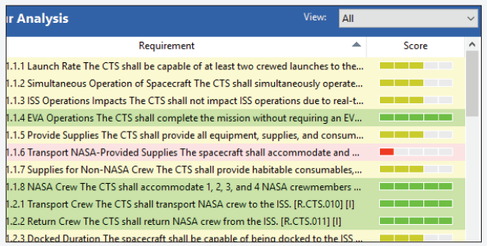
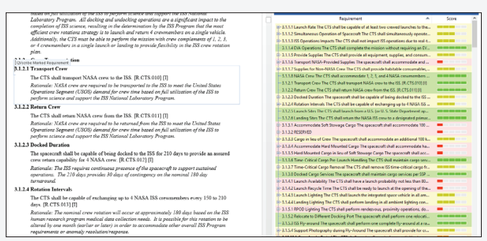


Figure 9: Highlighting of quality indicators following requirements analysis



THE BENEFITS OF NLP REQUIREMENTS ANALYSIS

This new generation of NLP requirements analysis tools will provide a number of important benefits to systems engineers and engineering project managers.

First, they automate a tedious, time-consuming, fatiguing and error-prone task, and accomplish it almost instantly. This not only saves time, it also saves domain experts from tasks that don't require domain knowledge. Again, this is not to say that expert review of requirements is unnecessary. Instead, these new tools streamline this task by immediately pointing out possible syntax problems, helping experts to correct such errors quickly and thus allowing them more time to focus on what really matters, like the semantics of the requirements or what requirements are missing.

Second, they instantly show where work is needed. The user can browse through the requirements scores and immediately see areas where the document is weak and perhaps needs extra attention, as well as which individual requirements need work.

Third, they prioritize users' revision tasks. Users can simply start with the lowest rated requirements (the one-star requirements in our QVscribe example) and work their way up towards the higher-rated ones.

Fourth, they are easily configured and optimized for any given domain and for changing user needs and preferences. Organizations can optimize these tools to their own policies and best practices, and users can configure them on the fly to adapt to specific situations and to test for specific quality indicators.

Fifth, NLP requirements analysis tools provide speed training for new systems engineers and project managers. Using such tools while authoring or analyzing requirements helps them quickly see mistakes they might be making, and helps them recognize those mistakes in others' work.

Sixth, these tools also help speed the authoring of high-quality requirements – even among experienced requirements engineers – by providing a sanity check of newly-written requirements, helping catch errors early and providing suggestions for improving those requirements on the fly.

Seventh, they speed review and editing of customer requirements specifications, helping requirements analysts and project managers catch problems, assess risk, and negotiate revisions, before they bid on projects and hand those customer requirements over to the system designers.

And finally, NLP requirements analysis tools help correct and eliminate requirements errors where they originate – during the requirements analysis and definition phase of the project – before they become more expensive to fix. For not only do these tools help detect and correct the half (according to Martinvi) of requirements defects that result from "poorly written, unclear, ambiguous or incorrect requirements." They also help realize additional savings by allowing domain experts more time to find those missing requirements that account for the other half of requirements errors.

CONCLUSIONS

Despite the rise of formal specifications and MBSE, the vast majority of requirements documents for complex systems are still written in natural language. They are thus vulnerable to errors due to the inherent ambiguities of natural language.

Since most errors in systems development originate in the requirements, and since the cost to fix errors increases in an exponential manner through successive phases of the project life cycle, it makes sense to try to catch requirements errors as early as possible – during the requirements definition phase of the project.

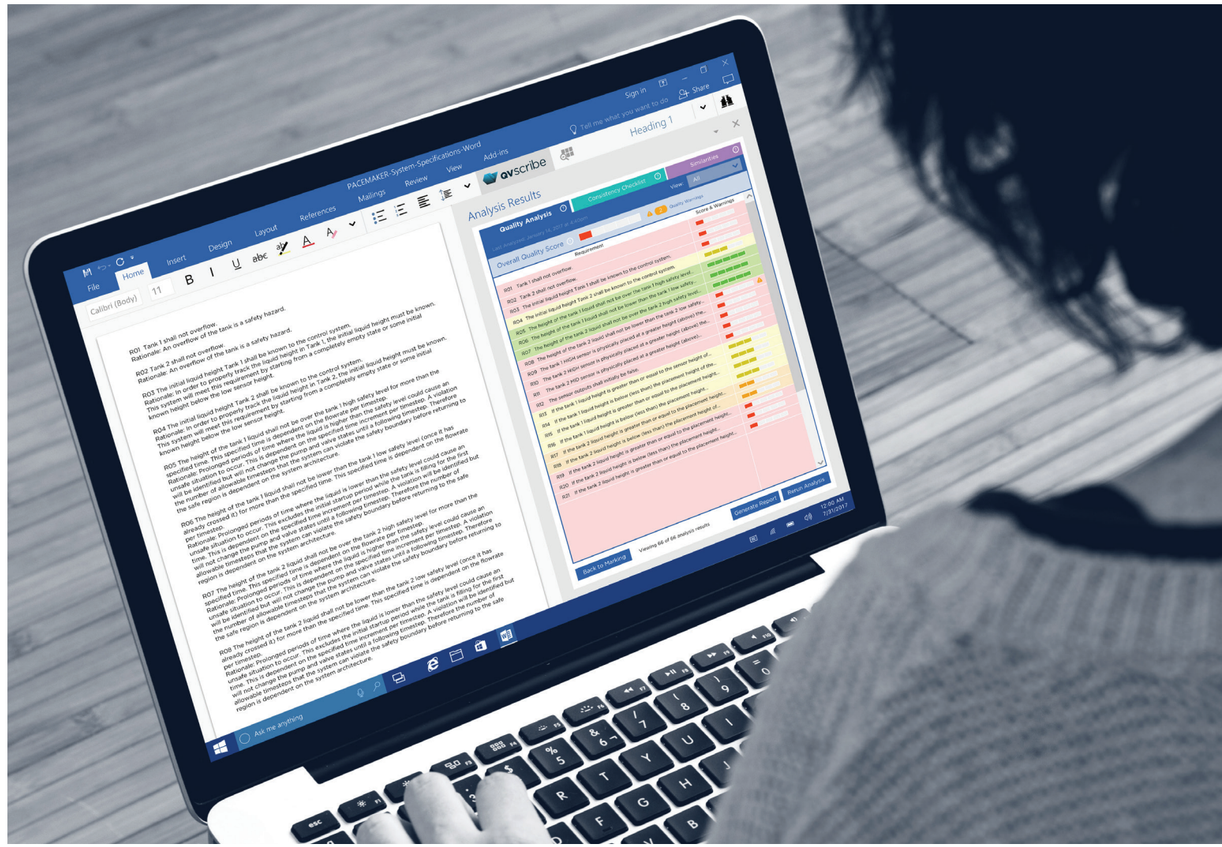
Until recently, finding errors in natural language requirements specificationshas been a labour-intensive proposition, relying primarily on the tedious, “brute force” techniques of checklist review and peer review.

But advances in natural language processing and the emergence of new NLP requirements analysis tools, like QVscribe, promise to streamline this process and allow domain experts to spend more of their valuable time and know how on what’s really important.

NEXT STEPS

QRA Corp has created one of the first of the emerging class of NLP requirements analysis tools described in this article. It’s called QVscribe. If you would like to learn more about QVscribe – or if you have an immediate need and would like to try QVscribe for yourself – visit the QVscribe product page at **qracorp.com/qvscribe/**.

While you’re there, sign up for our blog to be notified when the full release of QVscribe becomes available, and to get news and updates on QVscribe and other QRA products.



REFERENCES

i Jonette M.,et al, Error Cost Escalation Through the Project Life Cycle, NASA Johnson Space Center and INCOSE, June 2004.

ii Boehm, B. W., Software Engineering Economics, Prentice-Hall, 1981.

iii Martin, James, An Information Systems Manifesto, Prentice Hall, January 1984.

NB: While Martin's research is dated, we find it still cited in academic papers today, as there does not seem to be more recent study data on the subject. It also seems generally accepted among the systems engineers we've talked to that "at least half" of all system defects originate in the requirements. Given the strong emphasis on the build phase amongst quality tool suppliers, which Jones [x] (much more recently) points out, and IAG's finding [iv] that "68% of companies suffer from poor requirements specifications practices," we wonder if Martin's estimate might now be on the low side.

iv Ellis, Keith, Business Analysis Benchmark: The Impact of Business Requirements on the Success of Technology Projects, IAG Consulting, October 2008.

v Singh, S. and Saikia L., Ambiguity in Requirement Engineering Documents: Importance, Approaches to Measure and Detect, Challenges and Future Scope, IJARCSSE, October 2015.

vi Mich, Luisa, et al, Market research for requirements analysis using linguistic tools, Springer, Requirements Engineering, January 2004.

vii NASA Systems Engineering Handbook (Rev. 1), NASA, December 2007.

viii INCOSE Requirements Working Group, Guide for Writing Requirements, INCOSE, July 2015.

ix 21 Top Engineering Tips for Writing an Exceptionally Clear Requirements Document, QRA Corp, June 2016.

x Jones, Capers, Software Defect Origins and Removal Methods, Namcook Analytics, December 2012.

xi Wilson, H., Rosenberg, L., Hyatt, L., Automated Analysis of Requirement Specifications, ICSE, May 1997.

xii Kof, Leoinid, Natural Language Processing for Requirements Engineering: Applicability to Large Requirements Documents, Requirements Engineering, August 2004.

xiii Sateli, B, et al, Can Text Mining Assistants Help to Improve Requirements Specifications?, MUD, October 2012.

xiv Singh, S. and Saikia L., Ambiguity in Requirement Engineering Documents: Importance, Approaches to Measure and Detect, Challenges and Future Scope, IJARCSSE, October 2015.

xv Mich, Luisa, et al, Market research for requirements analysis using linguistic tools, Springer, Requirements Engineering, January 2004.

xvi NASA Systems Engineering Handbook (Rev. 1), NASA, December 2007.

xvii INCOSE Requirements Working Group, Guide for Writing Requirements, INCOSE, July 2015.

xviii 21 Top Engineering Tips for Writing an Exceptionally Clear Requirements Document, QRA Corp, June 2016.

xix Jones, Capers, Software Defect Origins and Removal Methods, Namcook Analytics, December 2012.

xx Wilson, H., Rosenberg, L., Hyatt, L., Automated Analysis of Requirement Specifications, ICSE, May 1997.

xxi Kof, Leoinid, Natural Language Processing for Requirements Engineering: Applicability to Large Requirements Documents, Requirements Engineering, August 2004.

xxii Sateli, B, et al, Can Text Mining Assistants Help to Improve Requirements Specifications?, MUD, October 2012.

version 1