Getting the right requirements right

Joseph Kasser
Visiting Associate Professor
Temasek Defence Systems Institute
National University of Singapore
Block E1, #05-05,
1 Engineering Drive 2 Singapore 117576
tdskj@nus.edu.sg, joseph.kasser@incose.org

Abstract: Research has shown that there is an on-going consensus that:

1. good requirements are critical to the success of a project,
2. the current requirements paradigm produces poorly written requirements and
3. ways of producing better requirements have been around for more than 20 years.

So, instead of producing yet another paper on how to write better requirements, this paper begins by posing the following question: why do systems and software engineers continue to produce poor requirements when ways to write good requirements have been documented in conference papers and textbooks? The paper then documents findings from research into the problem via the systems thinking perspectives and hypothesises that there are two requirements paradigms; the original A paradigm and the current B paradigm which is inherently flawed. The paper then dissolves the problem of poor requirements by applying technology to reduce the need for most of the requirements written today.

The perennial problem of poor requirements

Research has shown that there is an on-going consensus, typified by (Kasser and Schermerhorn, 1994; Jacobs, 1999; Carson, 2001; Hooks, 1994; DOD, 2011; Goldsmith, 2004; Wheatcraft, 2011; Lee and Park, 2004; Jorgensen, 1998) that:

1. good (Well-written) requirements are critical to the success of a project,
2. the current requirements paradigm produces poorly written requirements and
3. ways of producing better requirements have been around for more than 20 years.

So, instead of producing yet another paper on how to write better requirements, this paper begins by posing the following question: why do systems and software engineers continue to produce poor requirements when ways to write good requirements have been documented in conference papers and text books, e.g. (Jorgensen, 1998; Lee and Park, 2004; Alexander and Stevens, 2002)?

Situational analysis

This section analyses the situation from the systems thinking perspectives (Kasser and Mackley, 2008) to develop an answer to the question.

Big picture perspective

There seem to be a number of reasons for systems and software engineers to continue to produce poor requirements when ways to write good requirements have been documented in conference papers and text books, including (in no particular order):

- Lack of time to write the requirements due to schedule constraints, which results in poorly drafted and incomplete requirements.
- Failure of the stakeholders to articulate the requirements, which results in incomplete and sometimes results in incorrect requirements.
- Lack of training for writing good requirements sometimes due to budget issues, which results in poorly written requirements.
- Fundamental lack of understanding of the need for, and the purpose served by, requirements, by management, which results in lack of sufficient time for the requirements elicitation and elucidation process.
- Lack of implementation and solution domain knowledge in the systems and software engineers eliciting and elucidating the requirements, which tends to result in incomplete and sometimes unachievable requirements.
- Lack of functionality in commercial requirements tools that can call attention to requirements that are poorly written.

These reasons can be aggregated into two issues:

1. Production of poorly written requirements.
2. Lack of ways of ensuring completeness of the requirements.

Operational perspective

Requirements are written in the context of the system acquisition and design process which takes place in the front-end of the system development lifecycle (SDLC)[1]. Since various depictions of this process exist (Kasser and Hitchins, 2010), this paper uses the representation of the process shown in Figure 1[2] (Bahill and Dean, 1997) as a typical example. This version of the process begins with a customer request, which is then analysed, and a problem statement developed. Requirements, behavioural scenarios and models are then produced in parallel to develop a number of conceptual solution systems. After a feasible conceptual solution has been selected, the process to realize the solution system (for the following phases of the

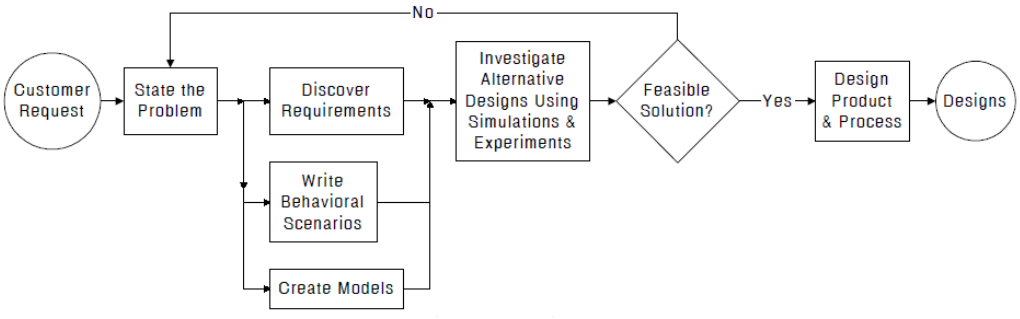

Figure 1 System Design Process (Bahill and Dean, 1997)

[1] The notion that the solution system is generally a technological system that needs to be developed, and hence the name system development lifecycle, seems to be US Department of Defense inspired. Essentially, there need not be any (technological) development; instead, solution systems can be synthesized by bringing together existing systems to create a new unitary whole.

[2] The figure seems to use the terms "solution" and design" interchangeably.

SDLC) is designed (Biemer and Sage, 2009), customized or architected (Kasser, 2005) and the SDLC continues.

Overlapping streams of work. One of the consequences of the focus on the functional and performance aspects of the solution system and the neglect of the non-functional attributes has been an increase in the complexity and cost of the acquisition process. This is due partly to the growth of the activities performed in Test and Evaluation (T&E), Integrated Logistics Support and Configuration Management to ensure that the systems acquired meet the needs of the user when fielded (irrespective of quality and completeness of the requirements). These activities now form parallel overlapping streams of work within the SDLC, generate nugatory work producing legally required documents that make little if any contribution to the success of the project while escalating the costs. Consider the following three examples of the overlaps:

- SEMP and TEMP
- Logistics Support Analysis
- Configuration Management

SEMP and TEMP. The contents of a Test and Evaluation Master Plan (TEMP) overlap the contents of a Systems Engineering Management Plan (SEMP) (Florida, 2006).

Logistics Support Analysis (LSA) is defined as an activity within Integrated Logistics Support which generates a Logistics Support Analysis Record. The activity is defined as "*the iterative process of identifying support requirements for a new system, especially in the early stages of system design"*. The main goals of LSA are to ensure that the system will perform as intended and to influence the design for supportability and affordability. LSA, performed as integral part of system design (up front):

- Produces supportability requirements as an integral part of system requirements and design.
- Defines support requirements that are optimally related to the design and to each other.
- Defines the required support during the operation phase of the system.

Configuration Management is defined as "*a field of management that focuses on establishing and maintaining consistency of a system's or product's performance and its functional and physical attributes with its requirements, design, and operational information throughout its life*" (MIL-HDBK-61A, 2001). There are two types of configuration audits within configuration management:

- **Functional configuration audits**– which ensure that functional and performance attributes of a configuration item are achieved, and
- **Physical configuration audits** - which ensure that a configuration item is installed in accordance with the requirements of its detailed design documentation.

Configuration audits can occur either at delivery or at the moment of effecting a change. These audits are commonly known as verification and validation or testing in the systems engineering community.

Functional perspective

The functions or activities performed to discover requirements within Figure 1 are elaborated in Figure 2. This process is complicated containing activities to:

(1) determine if requirements are feasible,

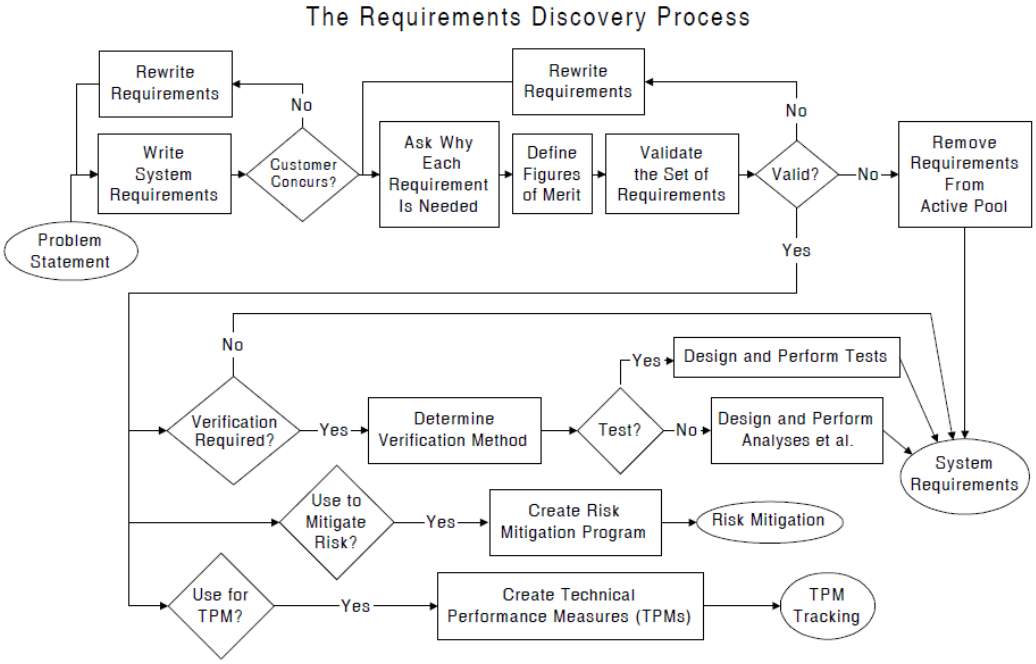

Figure 2 The Requirements Discovery Process (Bahill and Dean, 1997)

(2) identify risks associated with requirements, and
(3) detect correct and contradicting requirements in the step that asks why each requirement is needed.

However, the process cannot determine if the requirements are complete. As a result there is no way to show that the requirements express a complete solution that will meet the need until the solution system is actually fielded and put into service at the end of the SDLC, sometimes after many years and the expenditure of lots of money. As mentioned above, variations of this process exist. For example (Jorgensen, 1998) provides a variation on the process calling out the need for an evaluation of the operations concepts of a system before writing requirements.

The front end of the Australian SDLC is different to the example shown in Figure 1. There, an Operations Concept Document (OCD) is created and the functional and performance requirements for the system are based on the OCD (Gabb, et al., 2001). Gabb stated that an OCD may include identification and discussion of the following:

- Why the system is needed and an overview of the system itself.
- The full system life cycle from deployment through disposal.
- Different aspects of system use including operations, maintenance, support and disposal.
- The different classes of user, including operators, maintainers, supporters, and their skills and limitations.
- Other important stakeholders in the system.
- The environments in which the system is used and supported.
- The boundaries of the system and its interfaces and relationships with other systems and its environment.
- When the system will be used, and under what circumstances.
- How and how well the needed capability is currently being met (typically by existing systems).
- How the system will be used, including operations, maintenance and support.

The OCD is presented at an Operations Concept Review (OCR) and analysed in the subsequent phase of the SDLC and then used to create the functional and performance requirement documents.

Temporal perspective

From this perspective, one can examine how the need for requirements evolved and where requirements are produced and used in system acquisition and in the SDLC.

Research shows that the early systems engineers of the 1950's and 1960's tended to focus on identifying the problem (Wymore, 1993) and finding an optimal solution (Hall, 1962; Goode and Machol, 1959). These early systems engineers were Types III, IV, and V, (Kasser, et al., 2009) while the systems engineers who came later tended to focus on processes (Type II)'s. Back in the "good old days" of systems engineering, Type III, IV and V systems engineers remedied the problem in the early stage systems engineering activities addressing the conceptual solution. They then produced the matched set of specifications for the implementation or realization of the solution, and moved on to the next contract, leaving the Type II's to continue realizing the solution. There then came a time when there was a lack of new projects and so many of the Type III, IV and V's were laid off and lost to the discipline. When the need for systems engineers picked up again, in general, only the Type II systems engineers were left and they took over systems engineering. They had seen a successful process for developing systems following the production of the matched set of specifications and so their focus was on the post-requirements phases of the SDLC. They wrote the standards used in systems engineering (MIL-STD-499, 1969; MIL-STD-499A, 1974; EIA 632, 1994; IEEE 1220, 1998) for other Type II systems engineers to follow. As a consequence, the critical early stage engineering activities addressing the problem and conceptual solution were left out of mainstream Type II systems engineering (Bruno and Mar, 1997; Fisher, 1996). The Standards in turn became the foundation for educating systems engineers.

Quantitative perspective

While there is a consensus that requirements are critical, and that requirements suffer from several types of defects (see structural perspective), there is no widely accepted metric for the goodness of requirements (Kasser, et al., 2006) nor does there seem to be a widely accepted baseline definition of a requirement. For example the IEEE definition of a requirement is (IEEE 610, 1990):

(1) "A condition or capability needed by a user to solve a problem or achieve an objective.
(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
(3) A documented representation of a condition or capability as in (1) or (2)."

Yet variations of the definition continue to appear in the literature, including:

- Something that is wanted or needed, called for or demanded as being essential (Mason, et al., 1999).
- A statement which translates (or expresses) a need or constraints (technical, costs, times ...). (Fanmuy, 2004).
- Something obligatory or capabilities the system must satisfy (Powell and Buede, 2006).

- (Kossmann, et al., 2007) who cite a number of definitions in the literature and also provide a useful overview of the state-of-the-art of requirements engineering based on a wide collection of publications from previous years.

Structural perspective

A text-mode requirement should just be a simple sentence. Yet there are problems in the way requirement sentences are structured (Scott, et al., 2006). As discussed above, contemporary requirements management practice irrespective of the process used to generate the requirements is far from ideal, producing:

- **Vague and unverifiable requirements** – due to poor phrasing of the written text.
- **Incompletely articulated requirements** – due to a poor requirements elicitation process.
- **Incomplete requirements** – due to various factors including domain inexperience, and the lack of expertise in eliciting and writing requirements by technical staff.
- **Poor management of the effect of changing user needs during the time that the system is under construction** – due to lack of the understanding of the need for change management, and use of appropriate tools to do the function in an effective manner.

There also has been recognition that a requirement is more than just the imperative statement. For example, both (Alexander and Stevens, 2002) and (Hull, et al., 2002) discuss additional properties of the text-based requirement (e.g. priority and traceability) in conjunction with improving the writing of requirements. The IEEE Computer Society Computing Curriculum - Software Engineering --- Public Draft 1 --- (July 17, 2003) Software Engineering Education Knowledge Software expands on the earlier IEEE 610 definition of a requirement as follows:

> "*Requirements identify the purpose of a system and the contexts in which it will be used. Requirements act as the bridge between the real world needs of users, customers and other stakeholders affected by the system and the capabilities and opportunities afforded by software and computing technologies. The construction of requirements includes an analysis of the feasibility of the desired system, elicitation and analysis of stakeholders' needs, the creation of a precise description of what the system should and should not do along with any constraints on its operation and implementation, and the validation of this description or specification by the stakeholders. These requirements must then be managed to consistently evolve with the resulting system during its lifetime.*"

However, in practice, there is difficulty in adding these additional properties to the traditional requirement document or database and then managing them. This is because the current systems and software development paradigm generally divides the work in a project into three independent streams – Management, Development, and Test (Quality) (Kasser, 1995). Thus requirements engineering tools contain information related to the Development and Test streams (the requirements) while the additional properties tend to be separated in several different tools, (e.g. Requirements Management, Project Management, Work Breakdown Structures, Configuration Control, and Cost Estimation, etc.).

Generic perspective

Text mode requirements are but one way to communicate information. Other ways of communicating all or part of the same information include models, simulations, photographs, schematics, drawings, and prototypes. Fanmuy clarifies the definition of a requirement statement by adding "*this statement is written in a language which can take the form of a natural language or a mathematical, arithmetic, geometrical or graphical expression*" (Fanmuy, 2004). Timing and state diagrams are often used in Requirements Documents. Thus the con-

cept of stating user needs (under certain circumstances) via diagrams is already in use in systems engineering (Kasser, 2002b). Thus, from this perspective, requirements are but one of a number of communications tools. The focus should be on user needs, not on requirements. Van Gaasbeek and Martin quoted Dahlberg as stating, "*we don't perform system engineering to get requirements*" and, "*we perform system engineering to get systems that meet specific needs and expectations*" (Van Gaasbeek and Martin, 2001).

Continuum perspective

This perspective also looks at ranges and ambiguities. For example:

- The information in a requirements document can be looked at as both a solution and a problem. The matched set of specifications document a conceptual solution system that should remedy the problem. Thus, they document a solution as far as the customer is concerned but at the same time, they also document the problem faced by the designers who have to design the solution system.
- The word 'requirement' may have different meanings in different phases of the acquisition life cycle. In the early stage systems engineering activities in column A of the HKMF, 'requirement' and 'need' may also be used interchangeably, but have slightly different meanings in the worldviews of the customer and contractor. These differences in meanings show up in the IEEE definition of a requirement (IEEE 610, 1990).

Scientific perspective

There seem to be two requirements engineering paradigms.

The first paradigm begins with the systems engineering activities performed in column A the Hitchins-Kasser-Massie Framework (HKMF) for understanding systems engineering (Kasser, 2007) shown in Figure 3. The second paradigm skips the column A activities and begins in column B. For those readers not familiar with the HKMF, Phases A and B contain the following activities (Kasser, et al., 2009):

Layer of Systems Engineering \ Phase in the Life Cycle		Needs identification	Requirements	Design	Construction	Unit testing	Integration & testing	O&M, upgrading	Disposal
Socio-economic	5								
Supply Chain	4								
Business	3								
System	2								
Product	1								
		A	B	C	D	E	F	G	H

Figure 3 The HKM Framework for understanding systems engineering

A. **Identifying the need**. Phase A contains the early stage systems engineering activities addressing the problem and determining the conceptual solution and is based on (Hall, 1962), (Gelbwaks, 1967), (Hitchins, 1992) and the summary in (Brill, 1998). Phase A comprises the following sub-phases:

   A.1. This sub-phase contains the set of activities that explore/scope the problem, leading directly to Phase A.2. The activities performed in this phase produce a definitive statement of the problem-in-context.

   A.2. This sub-phase contains the set of activities that conceive the whole solution system (which 'emerges' from/"complements" the problem) and produces the concept of operations (CONOPS) that describes how the solution system will operate in its future environment.

   A.3. This sub-phase contains the set of activities that design the whole solution system, identify the environment, other interacting systems, the subsystems, parts, interactions, functional architecture, physical architecture, etc., etc., - but still all of the whole.

B. Requirements analysis. This phase addresses the physical solution and its implementation and contains the set of activities that specify the solution system as a full set of specifications for the whole and for the parts and their infrastructure, including the environment, Weltanschauung or paradigm that justifies them. If the specifications are in the form of text mode requirements, the output of this phase tends to be at the 'A' specification level (MIL-STD-490A, 1985). Requirements in the form of a matching set of system and subsystem specifications should be produced before the design phase of the SDLC (Hitchins, 2007) which typically commences after the System Requirements Review

The two requirements paradigms

Consider the two paradigms.

The A paradigm

The A paradigm beings with the systems engineering activities performed in column A in the HKMF. Research into the systems engineering literature found that successful projects such as the NASA Apollo program (Hitchins, 2007) and the LuZ SEGS-1 solar project (Kasser, 2008) were characterised by a common vision of the purpose and performance of the solution systems among the customers, users and developers; namely a paradigm that began in column A of the HKMF. Moreover, the common vision related to both the mission and support functions performed by the solution system. The research finding was supported outside the systems engineering literature by similar findings in the process improvement and Total Quality Control literature (Deming, 1993; Dolan, 2003). In addition, note that Business Process Reengineering creates and disseminates/communicates a 'to-be' model of the operation of the conceptual reengineered organisation before embarking on the change process.

Requirements are developed as an intermediate work product in the SDLC to provide formal communication between the stakeholders. Many informal situations do not need requirements when the vision is present. For example, during the Luz system development there was a need for a position sensor to report on the angle between the focus of the mirror and the horizon (Kasser, 2008). Alternative approaches to identifying the position of the mirror examined were (a) a pulse counting based sensor, (b) an analogue sensor based on a potentiometer and (c) an absolute position optical shaft encoder sensor. The optical shaft

encoder option was selected after due consideration. A search through the component catalogues quickly identified a commercial-off-the-shelf (COTS) Gray code[3] absolute position rotary encoder with an eight-bit parallel transistor-transistor-logic (TTL) interface. There was no need to write any requirements for the position sensor interface in this informal situation. The team had a concept of what the system was supposed to do; they had the (electronics) domain knowledge necessary to understand Gray code and the specifications on a TTL parallel interface, and got on with the interface design.

Since the A paradigm is characterized by a common vision of the purpose of the mission and support functions of solution systems among the customers, users and developers, the quality of the requirements tends to have little if any impact on the functionality of the solution system.

The B paradigm

Many systems and software engineers have been educated to consider the systems engineering activities in column B of the HKMF as the first phase of the systems engineering process. For example,

1. according to (Martin, 1997) page 95), (Eisner, 1997) page 9), (Wasson, 2006) page 60) and (DOD 5000.2-R, 2002), pages 83-84) requirements are one of the inputs to the 'systems engineering process';
2. in one postgraduate class at University of Maryland University College the instructor stated that systems engineering began for him when he received a requirements specification (Todaro, 1988).

While DOD 5000 does call out the 'analysis of possible alternatives' subset of activities performed in Phase A.2 of the HKMF (DOD 5000.2-R, 2002) pages 73-74), those activities are called out as part of the separate seemingly independent Cost as an Independent Variable (CAIV) process. CAIV is a way of complicating just a part of the concept of designing budget tolerant systems (Denzler and Kasser, 1995) using the cataract approach (Kasser, 2002a) and takes place **before** the DOD 5000.2-R 'systems engineering process' begins.

Discussion

The B paradigm is inherently flawed. This is because even if systems and software engineers working in a paradigm that begins in HKMF column B could write perfectly good requirements, they still cannot determine if the requirements and associated information are correct and complete because there is no reference for comparison to test the completeness. Consequently, efforts expended on producing better requirements have not, and will not, alleviate the situation. The situation cannot be alleviated because the situation is akin to participating in Deming's red bead experiment, which demonstrates that errors caused by workers operating in a process are caused by the system rather than the fault of the workers (Deming, 1993) page 158). Recognition that the B requirements paradigm is inherently flawed is not a new observation. For example:

- (Sutcliffe, et al., 1999) proposed reducing human error in producing requirements by analysing requirements using an approach of creating scenarios as threads of behaviour

[3] A binary code, where two successive values differ in only one bit, originally designed to prevent spurious outputs during transitions from one state to another in electromechanical switches.

through a *use case*, and adopting an object-oriented approach; namely they proposed a return to the A paradigm.

- Daniels et al. point out that standalone requirements make it difficult for people to understand the context and dependencies among the requirements, especially for large systems and suggest using use cases to define scenarios (Daniels, et al., 2005).
- One of the two underlying concepts of Model Based Systems Engineering (MBSE) is to develop a model of the system to allow various stakeholders to gain a better understanding of how well the conceptual system being modelled could remedy the problem, before starting to write the requirements. MBSE with its roots in the process camp of systems engineering and the B paradigm has discovered the CONOPS and is trying to return to the A paradigm.

Upgrading the A paradigm for the 21st century

In the second half of the 20th century the CONOPS, requirements specifications and associated information were stored in the form of separate documents containing text and graphics. In the latter years of the 20th century, information technology provided electronic storage capability in the form of databases. In the 21st century, the CONOPS and requirements can be linked via technology. Technology allows a multi-media approach to be used to communicate the vision of the solution system. This concept was described and prototyped as an Operations Concept Harbinger (OCH) which may be thought of as a multimedia OCD that also contains measures of effectiveness for each operational scenario (Kasser, et al., 2002). The OCH architecture consists of an underlying database and agents that act on the contents of the database to describe relationships and performance in various scenarios in the language of the viewer. Therefore, some designers could see the solution system execution expressed in UML, SysML, or IDEF0, and other stakeholders could see the solution as PowerPoint slides, videos, sound bites, etc. A prototype OCH was constructed and used in a successful Force Level Systems Engineering application (Kasser, et al., 2002).

Relating the CONOPS to Requirements

The use of *use case*s within a CONOPS in an object-oriented approach describing 'properties of', and 'services' (functions) provided by, components, can often provide the same representation of user needs as that of "requirements" if each property consists of an attribute and a value (Kasser, 2000). Consider "property" as the totality of the attribute and its value. Then 'requirements' can be stated as the properties and services needed, and capability (or functionality) can be stated as the value of the properties and services measured or exhibited by the component. The words functional and non-functional requirements no longer have to be used. When the system is broken down into subsystems, each property (attribute and value) is allocated to subsystem elements. Traceability of properties (functional and non-functional) is built into the approach.

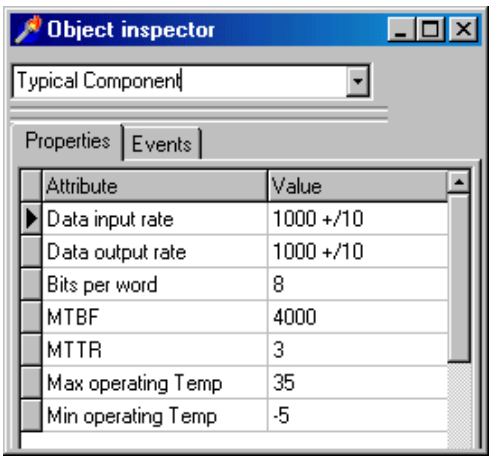

Figure 4 An objective view

The typical objective perspective of a component shown in Figure 4 shows the properties and services of a component. In the example of a communications component, the services performed by the object have to do with ingesting the data input, performing some action on the data and forwarding processed data. The data input attribute

has a value of 1000±10 units, the data output attribute a value of 1000±10 units. The figure also shows non-functional properties or attributes of the component such as reliability (expressed as MTBF, and MTTR) and operating temperatures. Other non-functional attributes such as colour and weight associated with a component can also be shown in the property viewer. The object-oriented approach also provides for inheritance of attributes of various classes of components which helps to maximise the completeness of the information in the CONOPS.

The first parts of the SDLC in the version according to Hitchins are shown in Figure 5 (Hitchins, 2007):Figure 6.2)[4]. The solution options may be constructed in the form of an OCH or an information-technology based CONOPS often called a model rather than in a document-based format.

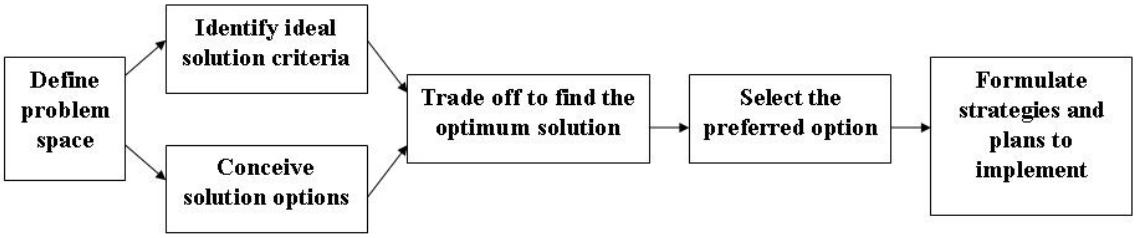

Figure 5 A systems engineering approach to problem solving (Hitchins, 2007)

So, if the model or an object-oriented CONOPS (OOCONOPS) can represent the user's needs in a manner verifiable by all stakeholders, there is no need for writing many of the requirements that seem to be needed in the B paradigm. The process shown in Figure 2 can be simplified. For example, there is no need to question the need for requirements since the OOCONOPS contains that information. In fact, since the OOCONOPS can communicate the vision more thoroughly than do text mode requirements, the number of requirements can also be reduced[5]. Consequently, instead of producing a mixture or requirements and behavioural models, systems and software engineers need to produce a single OOCONOPS for each of the solution system options. These are discussed with the customer and other stakeholders to verify that the selected solution system depicted in the OOCONOPS meets the need and will remedy the problem.

The object-oriented paradigm encapsulates processes (functionality) as well as data into an object. Consider the following types of 'smart' functions that might be encapsulated within an OOCONOPS:

- **Text clarification.** This function could scan the wording of any requirements and flag those requirements that are poorly written from the testing perspective. Research at the Systems Engineering and Evaluataion Centre at the Univeristy of south Australia developed a prototype software tool that performed this process (Kasser, 2002c). An updated tool to ingest and elucidate requirements (TIGER), shown in Figure 6, has been used in class lectures on requirements engineering in several postgraduate courses. Before TIGER was introduced, the discussions in the tutorials focussed on the structure and format of requirements. After TIGER was introduced and used to elucidate sample requirements, the focus of the in-class discussions changed to cover the difficulties of writing good requirements.

[4] Which unlike Figure 1 is abstract enough not to describe the format of the solution options.
[5] Note, in some cases, some requirements may still be needed at the contractual boundaries between contractors and sub-contractors.

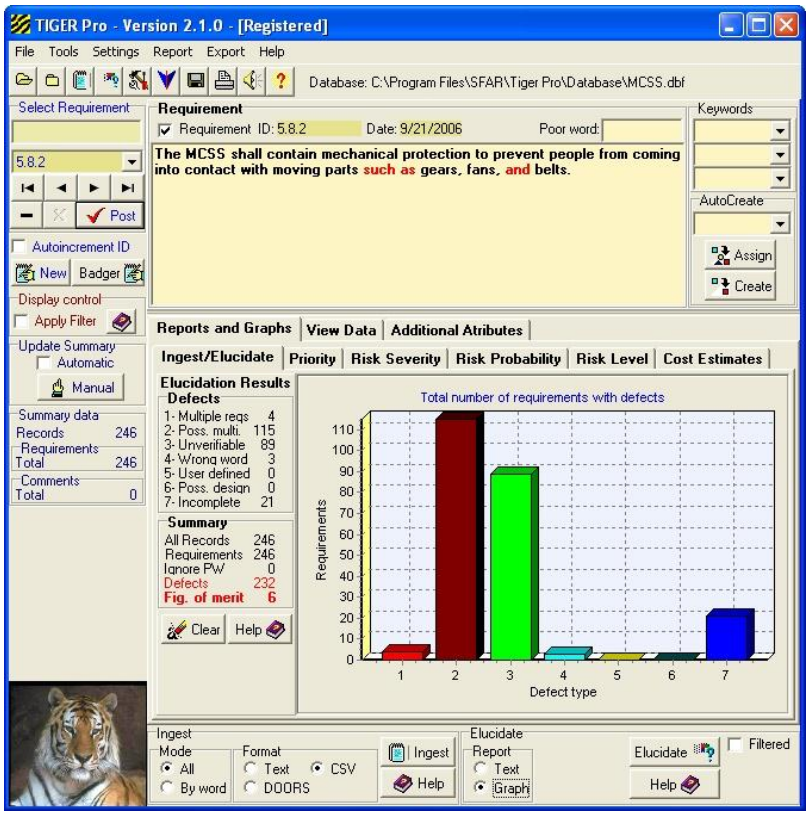

Figure 6 Tool to Ingest and Elucidate Requirements (TIGER)

- **Feasibility checker.** This function could check the feasibility of the scenarios and requirements and flag those that were not feasible at the time they were written before they were accepted.
- **Risk reduction and monitoring.** This function could ensure that all risks associated with scenarios and requirements have mitigating strategies and each strategy is implemented in a Work Package element, and provide appropriate warnings.
- **Property correlation.** This function could correlate properties and provide an indication when something needs further examination. For example, the function could correlate:
  - Estimated cost to implement with priority and provide an indication of requirements that have a high estimated cost to implement and a low priority. The customer could be asked if the requirements are really needed.
  - High risk with low priority. The customer could be asked if the requirements are really needed.
  - Text of requirement with acceptance criteria and identify requirements without corresponding acceptance criteria.

Inheritance is a major advantage of the object-oriented systems engineering paradigm since most new systems that are similar to, or can be considered as a class of, an existing system. For example, a communications satellite is a type of spacecraft, a destroyer is a type of surface ship, and a new car is similar (if not almost identical) to the previous model. Requirements reuse is becoming desirable. However, reuse is carried out in an ad-hoc manner in a document based paradigm (Von_Knethen, et al., 2002); the person in charge copies an old document and edits or enhances all parts they consider relevant. They integrate parts from other documents that deal with functionalities they have to add. Obviously, this approach is error prone. The concept of inheritance can be applied to scenario and requirement reuse to reduce errors. For example, inheritance could be implemented as:

- A function that identifies the type of system and inherits scenarios and requirements from a standard database for that type (class) of system. This functionality would maximize the completeness of the requirements by ensuring that applicable non-system specific performance requirements are not overlooked.
  - A function that allows selected scenarios and requirements to be copied from one database to another as demonstrated in TIGER for requirements. This functionality would save time when creating new systems having commonality with an existing system.
  - Templates containing selected information for specific types of document printouts from the database.
  - Standard templates for specific types of systems.

Discussion

The A paradigm is a return to the systems engineering paradigm of the good old days. In the A paradigm, the propensity for errors is much lower which provides an improvement in reliability and a reduction in the amount of work performed in the project (cost and schedule) as compared to the B paradigm which is inherently flawed. Getting the requirements right means minimizing the number of requirements and communicating the vision via an OOCONOPS. The remaining few requirements needed should be a printout from the OOCONOPS rather than being produced by a requirements analysis exercise. Today in some instances, requirements are bypassed when the contractor is asked to deleiver according to a schematic or copy a prototype. Hence, using an OOCONOPS is not really a completely new way of requirements engineering. Requirements are a means not an end.

Summary

The paper opened with the statement that research has shown that there is an on-going consensus that:

1. good requirements are critical to the success of a project,
2. the current requirements paradigm produces poorly written requirements and
3. ways of producing better requirements have been around for more than 20 years.

So, instead of producing yet another paper on how to write better requirements, this paper began by posing the following question: why do systems and software engineers continue to produce poor requirements when ways to write good requirements have been documented in conference papers and textbooks? The paper then documented findings from research into the problem via the systems thinking perspectives and hypothesised that there are two requirements paradigms; the original A paradigm and the current B paradigm which is inherently flawed. The paper then dissolved the problem of poor requirements by applying technology to reduce the need for most of the requirements written today.

Conclusion

The current requirements paradigm is inherently flawed and cannot be repaired. An OOCONOPS based paradigm can eliminate the need for many of the requirements necessary in the current requirements paradigm, increasing the probability of a successful project at lower cost. Accordingly, the OOCONOPS paadigm dissolves the problem of poor requirements by making most of the text-mode requirements used in the B paradigm irrelevant and unneccessary.

Biography

Joseph Kasser has been a practicing systems engineer for 40 years and an academic for about 14 years. He is a Fellow of the Institution of Engineering and Technology (IET), an INCOSE Fellow, the author of "*A Framework for Understanding Systems Engineering*" and "*Applying Total Quality Management to Systems Engineering*" and many INCOSE symposia papers. He is a recipient of NASA's Manned Space Flight Awareness Award (Silver Snoopy) for quality and technical excellence for performing and directing systems engineering and other awards. He holds a Doctor of Science in Engineering Management from The George Washington University. He is a Certified Manager and holds a Certified Membership of the Association for Learning Technology. He also started and served as the inaugural president of INCOSE Australia and served as a Region VI Representative to the INCOSE Member Board. He gave up his positions as a Deputy Director and DSTO Associate Research Professor at the Systems Engineering and Evaluation Centre at the University of South Australia in early 2007 to move to the UK to develop the world's first immersion course in systems engineering as a Leverhulme Visiting Professor at Cranfield University. He is currently a Visiting Associate Professor at the National University of Singapore.

References

Alexander, I. F. and Stevens, S., *Writing better requirements*, Addison-Wesley, 2002.

Bahill, A. T. and Dean, F. F., 1997, *The requirements discovery process*, proceedings of the 7th International Symposium of the INCOSE.

Biemer, S. M. and Sage, A. P., "Systems engineering: Basic concepts and life cycle," *Agent-directed simulation and systems engineering*, L. Yilmaz and T. Oren (Editors), Wiley-VCH, Weinheim, 2009.

Brill, J. H., *Systems engineering ---a retrospective view*, Systems Engineering 1 (1998), no. 4, 258-266.

Bruno, M. E. and Mar, B. W., 1997, *Management of the systems engineering discipline*, proceedings of the 7th Annual Symposium of the International Council on Systems Engineering.

Carson, R. S., 2001, *Keeping the focus during requirements analysis*, proceedings of the 11th International Symposium of the International Council on Systems Engineering.

Daniels, J., Bahill, T. and Botta, R., "A hybrid requirements capture process," *the 15th annual International Symposium of the INCOSE*, Rochester, NY., 2005.

Deming, W. E., *The new economics for industry, government, education*, MIT Center for Advanced Engineering Study, 1993.

Denzler, D. and Kasser, J. E., 1995, *Designing budget tolerant systems*, proceedings of the 5th Annual International Symposium of the NCOSE.

DOD 5000.2-R, "Mandatory procedures for major defense acquisition programs (mdaps) and major automated information system (mais) acquisition programs," US Department of Defense, 2002.

DOD, "National security space strategy unclassified summary," United States Department of Defense, 2011, p. 21.

Dolan, T., *Best practices in process improvement*, Quality Progress (2003), no. August 2003, 23-28.

EIA 632, "Eia 632 standard: Processes for engineering a system," 1994.

Eisner, H., *Essentials of project and systems engineering management*, John Wiley & Sons, Inc., New York, 1997.

Fanmuy, G., 2004, *Best practices for drawing up a requirements baseline - p192*, proceedings of the 14th International Symposium of the INCOSE.

Fisher, J., 1996, *Defining the roles and responsibilities of the systems engineering organization/team*, proceedings of the 6th Annual Symposium of the International Council on Systems Engineering.

Florida, "Writing a project systems engineering management plan," Florida Department of Transportation Traffic Engineering and Operations Office, 2006.

Gabb, A., Caple, G., Haines, D., Lamont, D., Davies, P., Hall, A., Van Gaasbeek, J., Eppig, S., Jones, D. and Vietinghoff, B., 2001, *Requirements categorization*, proceedings of the 11th Annual Symposium of the INCOSE.

Gelbwaks, N. L., *Afscm 375-5 as a methodology for system engineering*, Systems Science and Cybernetics, IEEE Transactions on 3 (1967), no. 1, 6-10.

Goldsmith, R. F., *Discovering real business requirements for software project success*, Artech House Inc., Boston, MA, 2004.

Goode, H. H. and Machol, R. E., *Systems engineering*, McGraw-Hill, 1959.

Hall, A. D., *A methodology for systems engineering*, D. Van Nostrand Company Inc., Princeton, NJ, 1962.

Hitchins, D. K., *Putting systems to work*, John Wiley & Sons, Chichester, England, 1992.

---, *Systems engineering. A 21st century systems methodology*, John Wiley & Sons Ltd., Chichester, England, 2007.

Hooks, I., 1994, *Writing good requirements*, proceedings of Proceedings of the 3rd NCOSE International Symposium.

Hull, M. E. C., Jackson, K. and Dick, A. J. J., *Requirements engineering*, Springer, 2002.

IEEE 610, "Ieee standard glossary of software engineering terminology," IEEE, 1990.

IEEE 1220, "Standard 1220 ieee standard for application and management of the systems engineering process," 1998.

Jacobs, S., 1999, *Introducing measurable quality requirements: A case study*, proceedings of the IEEE International Symposium on Requirements Engineering.

Jorgensen, R. W., 1998, *Untangling the twists in requirements analysis*, proceedings of the 8th INCOSE International Symposium.

Kasser, J. E., *Applying total quality management to systems engineering*, Artech House, Boston, 1995.

---, 2000, *Enhancing the role of test and evaluation in the acquisition process to increase the probability of the delivery of equipment that meets the needs of the users*, proceedings of The Systems Engineering Test and Evaluation Conference (SETE 2000).

---, 2002a, *The cataract methodology for systems and software acquisition*, proceedings of SETE 2002.

---, 2002b, *Does object-oriented system engineering eliminate the need for requirements?*, proceedings of the 12th International Symposium of the International Council on Systems Engineering.

---, 2002c, *A prototype tool for improving the wording of requirements*, proceedings of the 12th International Symposium of the INCOSE.

---, 2005, *Introducing the role of process architecting*, proceedings of The 15th International Symposium of the International Council on Systems Engineering (INCOSE).

---, *A framework for understanding systems engineering*, Booksurge Ltd, 2007.

Kasser, J. E., 2008, *Luz: From light to darkness: Lessons learned from the solar system*, proceedings of the18th INCOSE International Symposium.

Kasser, J. E., Cook, S. C., Scott, W., Clothier, J. and Chen, P., 2002, *Introducing a next generation computer enhanced systems engineering tool: The operations concept harbinger*, proceedings of SETE 2002.

Kasser, J. E. and Hitchins, D. K., 2010, *Unifying the different systems engineering processes*, proceedings of Conference on Systems Engineering Research.

Kasser, J. E., Hitchins, D. K. and Huynh, T. V., 2009, *Reengineering systems engineering*, proceedings of the 3rd Annual Asia-Pacific Conference on Systems Engineering (APCOSE).

Kasser, J. E. and Mackley, T., 2008, *Applying systems thinking and aligning it to systems engineering*, proceedings of the 18th INCOSE International Symposium.

Kasser, J. E. and Schermerhorn, R., 1994, *Determining metrics for systems engineering*, proceedings of The 4th Annual International Symposium of the NCOSE.

Kasser, J. E., Scott, W., Tran, X.-L. and Nesterov, S., 2006, *A proposed research programme for determining a metric for a good requirement*, proceedings of Conference on Systems Engineering Research.

Kossmann, M., Odeh, M., Gillies, A. and Ingamells, C., 2007, *'Tour d'horizon' in requirements engineering - areas left for exploration*, proceedings of the 17th International Symposium of the INCOSE.

Lee, J. Y. and Park, Y. W., 2004, *Requirement architecture framework (raf)*, proceedings of the 14th International Symposium of the INCOSE.

Martin, J. N., *Systems engineering guidebook: A process for developing systems and products*, CRC Press, 1997.

Mason, G. A., Sherwood, W. B., McGrath, W. E., Silva, F. G., Rutter, C. K. and Spence, J. P., 1999, *Application of system engineering principles in the development of the advanced photo system*, proceedings of the 9th INCOSE International Symposium.

MIL-HDBK-61A, *Military handbook: Configuration management guidance*, Department of Defense, 2001.

MIL-STD-490A, *Specification practices*, United States Department of Defense, 1985.

MIL-STD-499, *Mil-std-499 systems engineering management*, United States Department of Defense (USAF), 1969.

MIL-STD-499A, *Mil-std-499a engineering management*, United States Department of Defense (USAF), 1974.

Powell, R. A. and Buede, D., 2006, *Innovative systems engineering: A creative system development approach*, proceedings of the 16th International Symposium of the International Council on Systems Engineering (INCOSE).

Scott, W., Kasser, J. E. and Tran, X.-L., 2006, *Improving the structure and content of the requirement statement*, proceedings of The 16th International Symposium of the International Council on Systems Engineering (INCOSE).

Sutcliffe, A., Galliers, J. and Minocha, S., 1999, *Human errors and system requirements*, proceedings of IEEE International Symposium on Requirements Engineering.

Todaro, R. C., "Lecture handout, enee 648r," University of Maryland University College, 1988.

Van Gaasbeek, J. R. and Martin, J. N., 2001, *Getting to requirements: The w5h challenge*, proceedings of The 11th Annual Symposium of the INCOSE.

Von_Knethen, A., Paech, B., Kiedaisch, F. and Houdek, F., 2002, *Systematic requirements recycling through abstraction and traceability*, proceedings of IEEE Joint International Conference on Requirements Engineering.

Wasson, C. S., *System analysis, design, and development concepts, principles and practices*, Wiley-Interscience, Hoboken, New Jersey, 2006.

Wheatcraft, L. S., 2011, *Triple your chance of project success risk and requirements*, proceedings of the 21st Annual Symposium of the INCOSE.

Wymore, A. W., *Model-based systems engineering*, CRC Press, Boca Raton, 1993.