Object-Oriented Requirements Engineering and Management

Joseph E. Kasser DSc, CEng, CM, MIEE
Systems Engineering and Evaluation Centre
University of South Australia (UniSA)
Mawson Lakes
South Australia, 5095
Joseph.kasser@unisa.edu.au

Abstract

Object-Oriented requirements engineering is an approach to encapsulating information about the process and product, as well as functionality into a requirements object. This paper identifies properties of a requirement object based on information in the process (development, management and test and development streams of work in the system life cycle (SLC) as well as information about the product needed. The paper also describes some of the functionality that could be added to the requirements object. The paper concludes that Object-Oriented requirements engineering and management can effect a significant reduction of the problems currently encountered in the SLC due to poor requirements engineering and management.

Background

The systems and software development industry is characterized by a paradigm of project failure (Standish 1995). The situation has been described by Cobb's Paradox (Voyages 1996), which stated "*We know why projects fail, we know how to prevent their failure --so why do they still fail?"* One of the known contributing causes of these project failures is poor requirements engineering and management, which has been repeatedly and widely discussed and documented for at least 10 years (Hooks 1993; Kasser and Schermerhorn 1994; Jacobs 1999; Carson 2001; etc.). However, this continual documentation and discussion of the problem of poor requirements engineering and management has not resulted in a practical solution to the problem. This paper contains a preliminary introduction to an Object-Oriented approach to requirements engineering that might help to reduce the contribution of poor requirements engineering and management to project failures.

Requirements engineering and management

Requirements engineering and management is evolving. Dorfman and Thayer (1990) stated the definition of requirements engineering as "the science and discipline concerned with analysing and documenting requirements". Kotonya and Sommerville (1998) restated the definition of requirements engineering as "the systematic process of eliciting, understanding, analysing, documenting (and managing) requirements". In effect, they expanded the definition to include the elicitation process as well as the managing process. There has since been recent 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 states

> *"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.). Moreover, activities that have become increasingly identified as being critical to success (e.g. risk planning and management) in many cases are not performed in the current paradigm. In those cases where they are performed, they tend to be treated as add-on's, and implemented in a complex process that could have been designed and pictured by W. Heath Robinson (UK) or Rube Goldberg (USA).

The Object-Oriented paradigm

The Object-Oriented paradigm however, provides for these add-on activities and provides a place to store them, namely as properties within the requirement object. Consider "property" as the totality of the attribute and its value. Kasser (2000a) expressed "requirements" in terms of "properties and services needed" and capability in terms of "properties and services provided" where each property consists of an attribute and a value. 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 appropriate subsystem elements. Traceability of properties (functional and non-functional) is built into the approach. Moreover, as described below, the packaging of processes or functionality together with the data inside the requirement object provides for automating some of the manual processes that contribute to the successful realisation of systems but which may be overlooked in the current paradigm.

Object-Oriented systems engineering

The system life cycle (SLC) tends to begin with a problem or a need. The development process begins by clarifying the need and then articulating the need as a high level solution, in the form of a concept of operations of what the solution to the problem or type of system that satisfies the need, will do. Once the functionality of the system is understood, a set of requirements for a product (the system) that will meet the need is developed. This is reviewed at the system requirements review, and when accepted, is implemented to realise the needed solution to the problem. Thus, the focus of the requirement in the functional paradigm is on the properties and functionality of the product to be produced.

Object-Oriented systems engineering has a disconnection in its process as shown in Figure 1. Concepts of operations are stated in the form of Use Cases involving the interaction of

objects, and the system is developed to implement the Use Cases via an Object-Oriented approach.

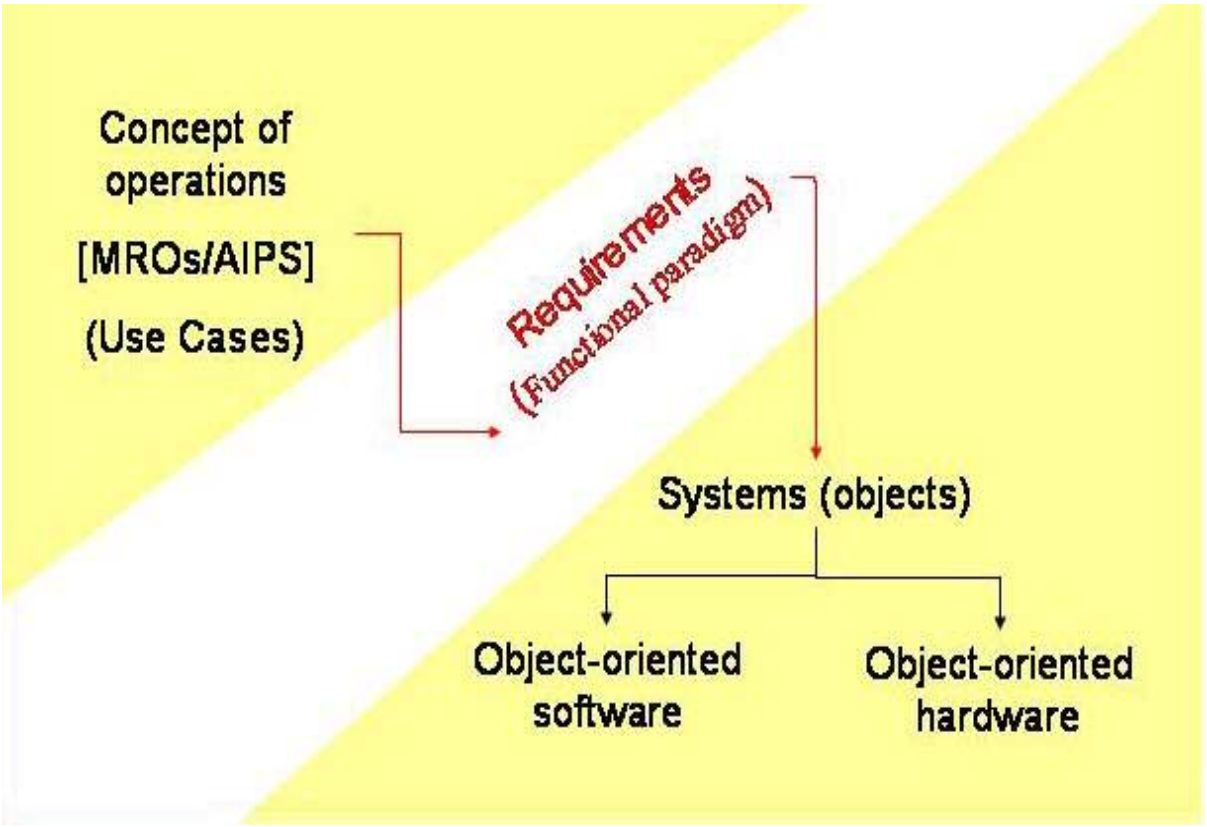

Figure 1 Object-Oriented Systems Engineering

Requirements however remain firmly in the functional paradigm as Object-Oriented systems and software engineering in general, has not, applied the object concept to requirements. The general approach seems to be to either

- Treat requirements as something that have to be produced in the early stages of the SLC, and sometimes, adding properties of traceability and priority. Schach (2002) for example, still partitions requirements into functional and non-functional, but does add the properties of traceability and priority.
- Ignore requirements other than those that can be expressed in Use Cases. This approach in general just seems to articulate the user's needs in two redundant ways (Use Cases and requirements).

Several approaches to Object-Oriented systems engineering have been proposed (e.g. Hopkins 1998; Meilich 1999; Lykins 2000) and there is an INCOSE working group working on upgrading the UML to add "systems" aspects.

So from a process, perspective the front end of the process (concept of operations stating the customer's need) is object-oriented, the back end of the process (the implementation of the system to meet the customer's needs) is also Object-Oriented but the requirements remain in the functional paradigm. Kasser and Schermerhorn (1994) wrote that systems engineers needed to use a methodology that seamlessly interfaced to the software development methodology to avoid delays and errors in translation from the system requirements to the

design phase of the SLC. Consequently, there is a need for Object-Oriented requirements in an Object-Oriented implementation paradigm.

Research Question

The major question is "what are the properties of an Object-Oriented requirement? The answer is not simple. The first avenue to be explored on this journey to identify the properties is the usage of requirements in the SLC and explore how Object-Oriented paradigm can improve the current situation.

Requirements drive the work

The requirements elicitation process produces a set of requirements, which represent the documentation of a product that meets the customer's needs. Consequently, every element of the work ought to be traceable (and chargeable) to a specific requirement. The work takes place in three streams of activities (management, test and evaluation, and development) (Kasser 1995); hence, every requirement can be thought of as having properties driving the work in each of the three streams. This produces a view of a requirement statement as the tip of an iceberg, where the statement can be seen, but the underlying work to produce the capability that meets the requirement is hidden. An alternative view more traditional view in the form of an overview of the documentation tree, is presented in Figure 2.

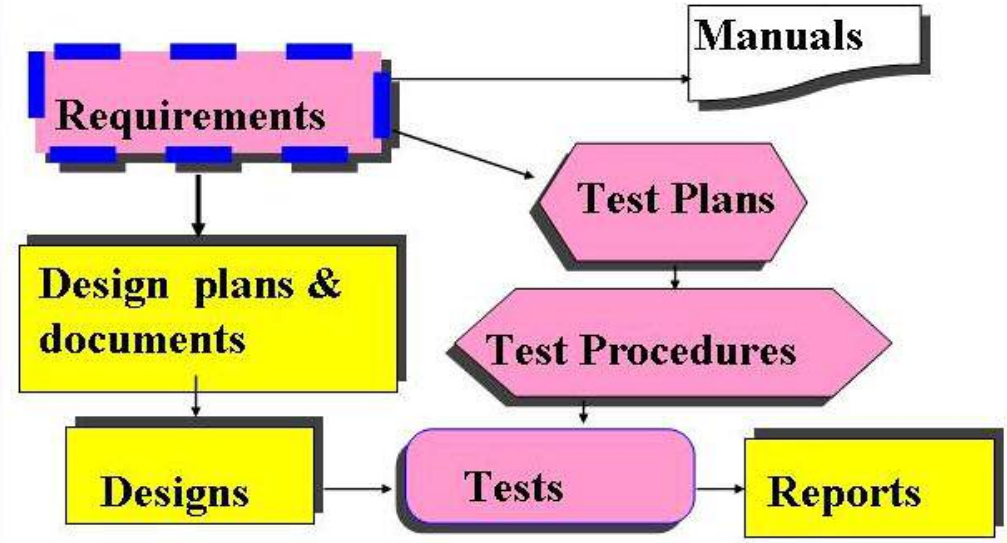

Figure 2 Requirements Drive the Work

Effectively, Object-Oriented requirements engineering and management not only performs the requirements engineering at the front end of the SLC, but also provides integrated information for the functions of project management, design, development, and test and evaluation, as performed in the current paradigm. As in the current paradigm, the implementation work plan can be published in three documents, namely:

- The system requirements document
- The systems engineering test and evaluation plan
- The system engineering management plan

Kasser (2000) researched the SLC from an information perspective, and determined that these documents may be considered as summaries of the properties of the requirements in each of the appropriate stream of work. Consider the contents of each of the documents. For

each document, the following properties of the requirement apply.

- The unique identification number – to clearly identify the requirement.
- The text of the requirement statement – conforming to the requirements for writing requirements (Kasser 1995).

The other properties of the requirements are document specific as follows:

The system requirements document

The system requirements document contains the documented solution of what has to be done to provide a solution to the customer's problem (based on the concept of operations). The system requirement document should contain the following information for each requirement:

- **Traceability to source(s)** – i.e. the concept of operations, regulations, people, etc.
- **Rationale for requirement** – to communicate the reason why the requirement was included in the first place. This information is important for considering change requests during the operations and maintenance phase of the system's life cycle. This information is sometimes included as comments in the current paradigm, but is not required.
- **Traceability sideways to other documents** (or databases) at the same level of decomposition of the system. This provides information for use by the configuration control board in considering the impact of requested changes.

The systems engineering test plan

The systems engineering test plan drives the testing and evaluation process. The systems engineering test plan should contain the following information for each requirement:

- **Acceptance criteria** – which are provided in response to the question "How will we know that the requirement has been met by the system?"
- **Planned verification methodology(s)** - demonstration, analysis, etc.
- **Testing parameters** – the sections of the test plans and procedures that verify the system meets the requirement.
- **Resources needed for the tests** – people, equipment, time, etc.

The systems engineering management plan

The systems engineering management plan contains the planned resources and schedule necessary to perform the design and testing activities. The systems engineering management plan should contain the following information for each requirement:

- **Traceability to implementation** – identifies the Build in which the requirement is scheduled to be implemented.
- **The priority of the require**ment.
- **The estimated cost** to construct and test the elements of the system that provided the functionality specified by the requirement.
- **The level of confidence in the cost estimate**.
- **Risks** – implementation, programmatic, and any other identified.
- **Production parameters** – the Work Breakdown Structure (WBS) for the work to be done to meet the requirement.
- **Required resources** for the work

Kasser (2000) summarised this information in the form of a set of Quality System Elements (QSE) as being necessary for effective system and software development. The QSE are:

- **Unique identification number** - the key to tracking.
- **Requirement -** the imperative construct statement in the text mode, or other form of representation.
- **Traceability to source(s) -** the business goals, previous level in the production sequence, or external source as appropriate.
- **Traceability to implementation -** the next level in the production sequence. Thus, requirements are linked to design elements, which are linked to code elements.
- **Priority -** knowing the priority allows the high priority items to be assigned to early Builds, and simplifies the analysis of the effect of budget cuts.
- **Estimated cost and schedule -** these feed into the management plan and are refined as the project passes through the SLC.
- **The level of confidence in the cost and schedule estimates** - these should improve as the project passes through the SLC.
- **Rationale for requirement** - the extrinsic background information and other reasons for the requirement.
- **Planned verification methodology(s) -** The customer's answer to the question "how will we know that the system meets this requirement?" provides the acceptance criteria for the requirement, which are then used in developing the planned verification methodology. Documenting this at the same time as the requirement avoids accepting requirements that are either impossible or too expensive to verify.
- **Risk -** any (implementation and programmatic) risk factors associated with the requirement.
- **Keywords -** allow for searches through the database when assessing the impact of changes.
- **Production parameters -** the Work Breakdown Structure (WBS) elements in the Builds in which the requirements are scheduled to be implemented.
- **Testing parameters -** the Test Plans and Procedures in which the requirements are scheduled to be verified. Test results are also included in this property.
- **Traceability sideways to document duplicate links -** required when applying the QSE to an existing paper based project.

The QSE can be considered as properties of requirements and thus at least improve on the current paradigm by providing a place to store those additional properties (by definition) in the form of a QSE database, an integrated database containing information about the process and product. Since the information in the QSE database covers the three streams of work (management, design, and test and evaluation) in the process as shown in Figure 3, the three streams of work are considered to be interdependent not independent (Kasser 1995).

Adding other object–oriented properties and processes to the QSE

Software engineering articulated the Object-Oriented approach as a way of encapsulating data and processes in ways that were not tied to physical implementations. Consider the addition of other Object-Oriented properties and processes to the data in the QSE database as follows.

Properties

The following properties could be added to the original QSE:

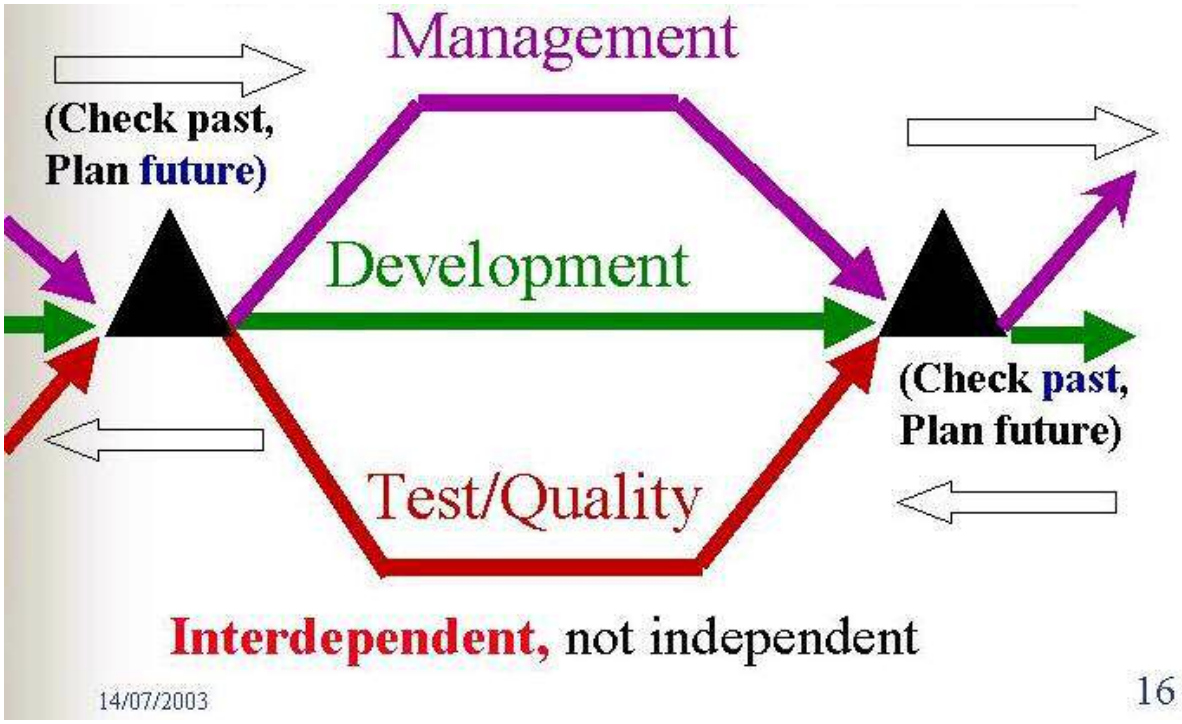

Figure 3 The Three Streams of Work building System

- **Non-functional elements of capability needed** – survivability, reliability, maintainability, etc.
- **Access control property**- to control access to the requirement or selected properties. This might be used in classified situations or in corporate situations where two companies share information.
- **Risk mitigation** – the strategies for mitigating the identified risks. This property has links to the WBS activities.

Processes

The Object-Oriented paradigm encapsulates processes (functionality) as well as data into an object. Consider the following types of processes that might be encapsulated within the QSE database implementation of an Object-Oriented requirement.

- **Text clarification.** This process could scan the wording of the requirement and flag any requirements that are poorly written or do not comply with the requirements for writing requirements (Kasser 1995). Kasser (2002) described a prototype software tool that performed this process. An updated tool to ingest and elucidate requirements (TIGER), shown in Figure 4, has been used in a class lecture on requirements engineering in three 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. This is a significant shift in perspective (Kasser et al., 2003).
- **Feasibility checker.** This process could check the feasibility of the requirements and flag those that were not feasible at the time they were written before they were accepted.
- **Risk reduction and monitoring.** This process could ensure that all risks have mitigating strategies and each strategy is implemented in a WBS element, and provide appropriate warnings.

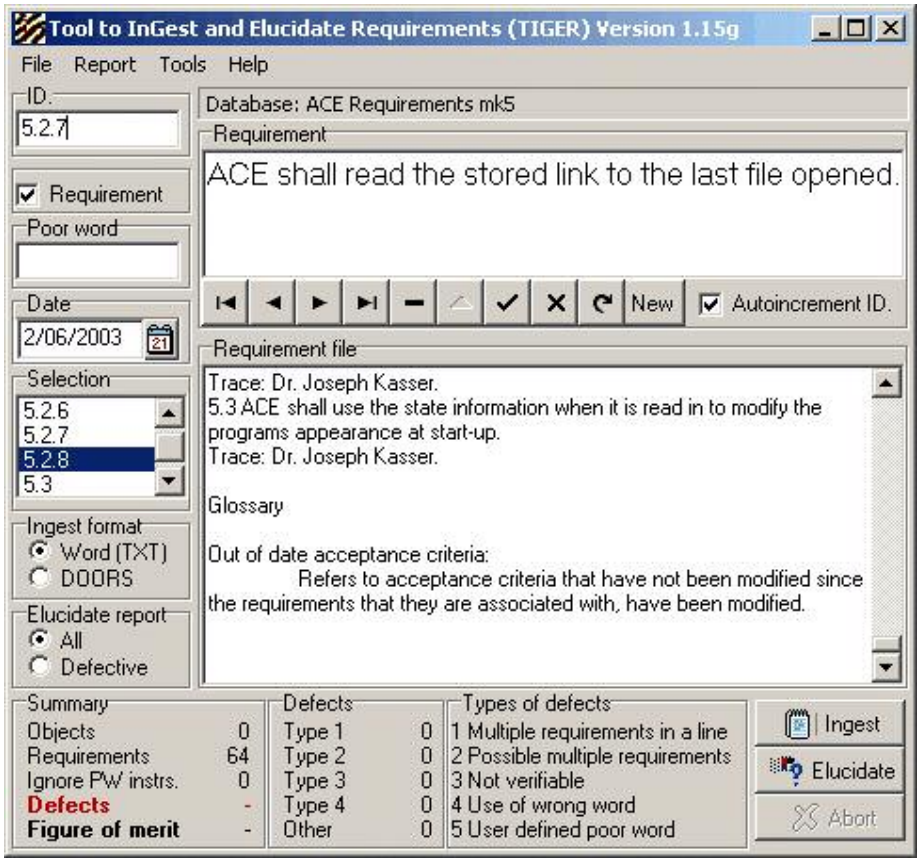

Figure 4 Tool to Ingest and Elucidate Requirements (TIGER)

- **Property correlation.** This process could correlate properties and provide an indication when something needs further examination. For example, the process 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.
- **Production process correctness**. This process could check for the presence or absence of other properties and provide indicators. For example, it could check that all requirements have acceptance, criteria, priorities, and cost estimates.
- **Progress monitoring.** The question "what do you mean, you can't tell me how much of my project has been completed?" is a very difficult one to answer in the current paradigm (Kasser 1997). However, the use of Categorized Requirements in Process (CRIP) charts proposed by Kasser (1997) can provide a better answer to the question than the measurements made in the current paradigm. They can also provide early identification of anomalies in the implementation process.
- **Facilitating and ensuring the completeness of testing**. This process could automate a manual process of building test compliance matricis. It would convert written requirement paragraphs containing multiple requirements into separate requirement paragraphs. As an example of the work that this tool could expedite, consider the following requirement (ST-DADS 1992):

204.1 ADS shall automatically maintain statistics concerning the number of times and the most recent time that each data set has been accessed. These same statistics shall be maintained for each piece of media in the DADS archive.

The function would split this requirement into the following four requirements to simplify tracking the completeness of the test plans:

204.1a DADS shall automatically maintain statistics concerning the number of times ~~and the most recent time~~ that each data set has been accessed. ~~These same statistics shall be maintained for each piece of media in the DADS archive~~.

204.1b DADS shall automatically maintain statistics concerning ~~the number of times and~~ the most recent time that each data set has been accessed. ~~These same statistics shall be maintained for each piece of media in the DADS archive.~~

204.1c DADS shall automatically maintain statistics concerning the number of times ~~and the most recent time~~ that ~~each data set has been accessed. These same statistics shall be maintained for~~ each piece of media in the DADS archive [has been accessed].

204.1d DADS shall automatically maintain statistics concerning ~~the number of times and~~ the most recent time that ~~each data set has been accessed. These same statistics shall be maintained for~~ each piece of media in the DADS archive [has been accessed].

Leaving the sections of the requirement that were not being tested in place but stricken through would clearly identify which section of the requirement is being tested. An unfortunate side effect is that it would also clearly show the defects in the requirement. Note that the phrase 'has been accessed' has been moved in the last two sub-requirements to clarify the sub-requirement.

Applying the concept of Inheritance

Inheritance may be used in ways that extend the software engineering usage. Traceability is an inheritance function. Design elements may be traced back to requirements, regulations, etc. Expert system technology could be used to build "human experience" and tacit knowledge into the process part of the object. This could add Artificial Intelligence to current generation requirements engineering and management tools.

Inheritance is a major advantage of the Object-Oriented Requirements Engineering paradigm since most requirements are written for systems that are similar to, or 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 currently carried out in an ad-hoc manner (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 requirement reuse to reduce errors. For example, inheritance could be implemented as:

- A function that identifies the type of system and inherits 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 requirements to be copied from one database to another. This functionality would save time when creating requirements for new systems having commonality of requirements with an existing system.
- Templates containing selected information for specific types of document printouts from the database.

Populating the properties of the requirement

The process of accepting requirements may be represented as shown in Figure 5 (Kasser 2002b). The customer's need (source of the requirement) is represented as a request for capability (requested requirement) and allocated an identification number. The requirement request is then assessed for priority, and cost and schedule impact on the SLC, as well as for risks and conflicts with existing requirements. The result of the impact assessment is presented to the customer who then decides to accept, reject, or modify the requirement request. However, some of these impact assessments are generally not performed in the current paradigm.

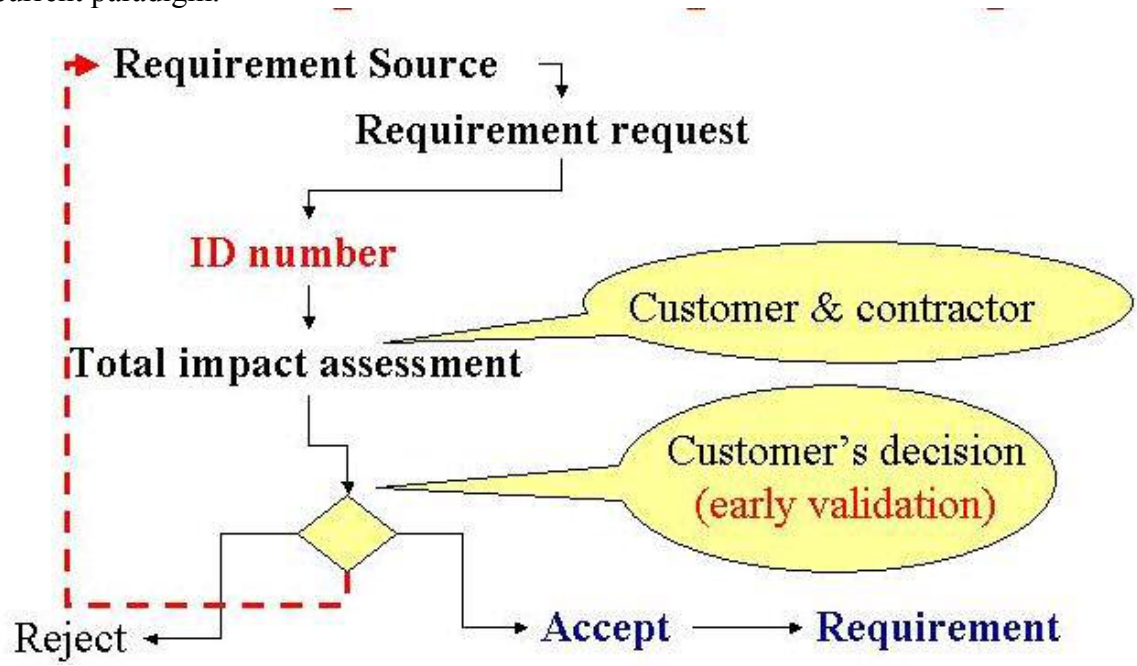

Figure 5 Process for generating requirements

The entire set of properties cannot be populated at the same time. Population begins as shown in Figure 6. The initial set of properties of a requirement request consists of the:

- Requirement statement or representation;
- Source of the requirement (traceability);
- Key words;
- Rationale for the requirement;
- Acceptance criteria,;
- Non-functional properties (eg. reliability, maintainability and survivability); and
- Priority of the need for the capability.

The requirement request is allocated an identification number. After performing the total impact assessment, the next set of properties are populated (Estimated cost to implement, Risk (implementation and programmatic) and the Build in which the requirement will be implemented. The impact assessment is then negotiated with the customer who may accept,

reject or modify the assessment. If accepted the properties are incorporated in the QSE database. Later on in the SLC, the test properties are populated as the test team develops the test plans etc. When the Build Plan is developed the implementation properties are further populated, and so on.

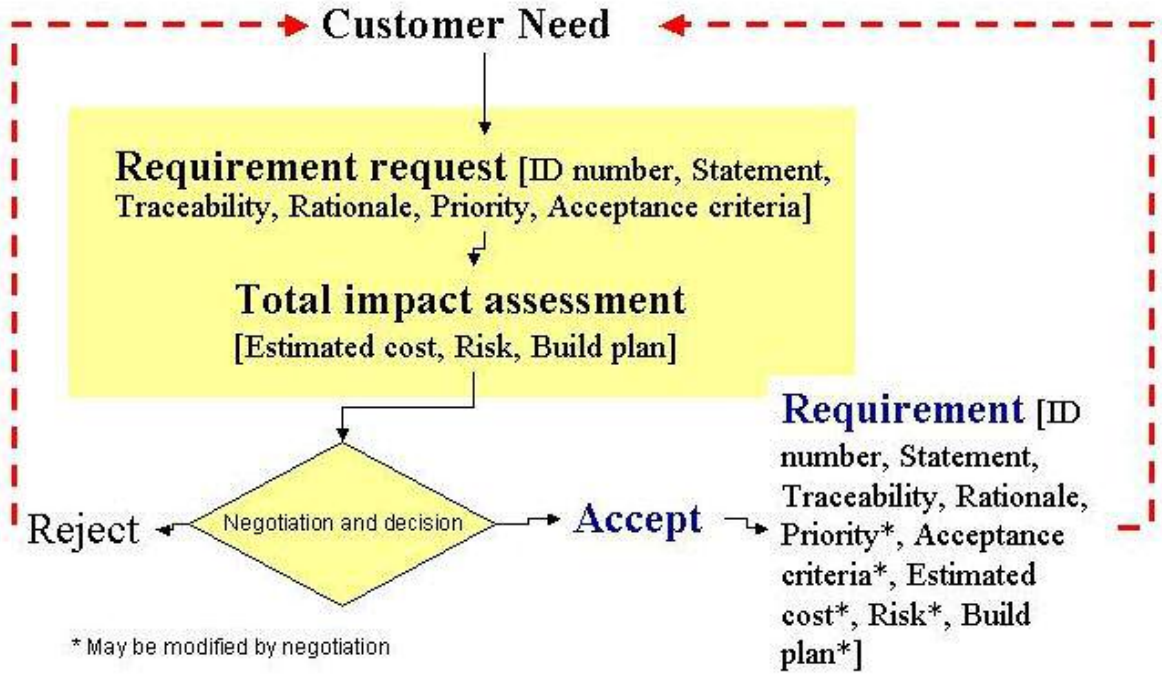

Figure 6 Populating the properties of the requirement

Tools research

Tools that allow the QSE to be defined and populated in an integrated manner do not typically exist at present. The current paradigm tools only focus on one the three streams work. Thus requirements engineering tools focus on the design and development stream, project management tools focus on the management stream, etc. The tools for an Object-Oriented requirements engineering and management paradigm (containing information and processes (artificial intelligence)) form the focus of the Prototype Educational Tools for Systems and Software (PETS) engineering research project at the Systems Engineering and Evaluation Centre (SEEC) at the University of South Australia (Kasser et al., 2003).

Other tools that have been developed within the framework of the Object-Oriented approach to requirements engineering are:

- **The Communications Requirements Evaluation & Assessment Prototype (CREAP) (**Kasser and Cook 2002) is targeted for use in a specify-to-inventory situation. It assists the writers of requirements by directing them to only specify requirements that are feasible in a given concept of operations. CREAP employs a frame-based approach (Cook et al. 2001) to reason about the feasibility of the requirements. CREAP is an Object-Oriented frame-based requirements engineering tool (FBRET) that integrates military communications domain expertise and requirements engineering practice. Specifically, it ensures that the set of equipment selected to meet the requirements imposed on a communications systems for a rapid force deployment will constitute a viable military information system.

- **The Operations Concept Harbinger (OCH)** (Kasser, Cook, et al., 2002) is a tool for capturing user needs without the explicit use of "text-mode requirements". It may be thought of as a multimedia Operations Concept Document that also contains measures of effectiveness for each operational scenario. The OCH helps to bridge the gap between the soft systems methodologies used in the early phases of the system development life cycle and the hard systems methodologies used in the construction of a system.

Benefits of the Object-Oriented approach to requirements engineering

Thus the Object-Oriented approach to requirements engineering and management

- Simplifies the SLC by containing the add-on tasks (e.g. risk management) in the current paradigm, by definition, which if implemented at all, tend to be implemented in a complex process that could have been designed and pictured by W. Heath Robinson (UK) or Rube Goldberg (USA).
- Improves the production of well-written requirements via the use of a tool to ingest and elucidate requirements (TIGER). A requirements workshop was held as part of postgraduate courses in Software Engineering at University of Maryland University College (UMUC). The workshop:
  - Discussed the problems resulting from poor requirements;
  - Provided examples of poorly written requirements;
  - Provided a set of requirements for writing requirements (Kasser 1995);
  - Asked the students to read and evaluate an instructor-supplied requirements document to determine the number of good and bad requirements in the document.

  The focus of the discussion in the workshop was on the structure of a requirement. When the TIGER tool was introduced into a similar workshop in three post-graduate classes at SEEC, the focus of the workshop discussion changed from the discussion of the structure of a good requirement (pre TIGER) to the discussion of the difficulty of writing a good requirement (post TIGER). This is a significant shift in perspective (Kasser et al., 2003)
- Links all work back to the original requirements or to design decisions based on the requirement. The three documents discussed above are more complete since the additional properties of the requirements provide information that is generally missing in the current paradigm.
- Provides early identification of anomalies in the implementation process via CRIP charts (Kasser 1997).
- Incorporates by definition the additional properties of requirements now becoming associated with requirements engineering and management, thus avoiding the "add-on" approach of the current paradigm.
- Provides more of the information necessary for effective command and control (management) of the resources needed to realize the system than is generally available in the current paradigm.
- Provides new perspectives on the system. For example, various properties can be examined at the System Requirements Review (before the system is built and the cost to effect changes is relatively low) and questions posed that are not easily identified or answered in the current paradigm. Typical examples are:
  - How will the customer know that a requirement has been met? The documentation of the requirement statement and acceptance criteria at the same time helps both

elucidate the requirement and minimize the acceptance of requirements whose implementation cannot be verified or achieved.
    - Is a high (estimated) cost to implement, low priority requirement really needed?
    - Have the high priority requirements been assigned to early Builds? This is desirable, so that future budget cuts would tend to eliminate the lower priority performance characteristics.
    - What is an appropriate Risk Profile for the type of system to be realised and is that the same profile as the instance being realised? The risks could be assigned values between 1 and 10, and the resulting profile presented as a Pareto chart to provide a visible risk profile. The Risk Mitigation plan would become another abstracted view of the QSE database.
- Minimises loss of information and maximises correctness of information across the SLC stages by considering all documents as views of the Object-Oriented (QSE) database.
- Provides for intelligence to assist in the elicitation and elucidation of requirements.
- When coupled with the Blackboard approach used in Artificial Intelligence, allows concept demonstration tools to be developed and deployed rapidly (Kasser and Cook 2003; Kasser et al., 2003).

Future research

The QSE database is not the properties of a requirements object. It is only the first few steps along a journey that will determine a set, or sets, of properties for Object-Oriented requirements. Future research will examine areas, such as which properties are appropriate, identifying classes and methods appropriate to the process properties as well as the product properties and the nature of the Requirements Class Hierarchy. This paper has provided a glimpse of what "might be". The SEEC PETS are planned as tools to facilitate this research. They will be used to work with the QSE in the manner described by Kasser and Cook (2003).

The other avenue of research will, in the manner of systems engineering, begin with a concept of operation for how the objects could be used. Typical use cases might start with the examples of process functionality mentioned above, and continue to explore:

- How the properties of Object-Oriented requirements might be used as smart checklists.
- How requirements objects can be typed, interrelated, and re-factored, as understanding of the design implications evolve.
- How requirements objects can be elaborated by Design and Test as the project progresses, thereby generating increased confidence of operational acceptance of the system.
- If and how tools such as the PETS that incorporate requirements objects may be used as the high-tech equivalent of the paper-based Military Standards, so beloved of the previous generation of systems engineers.
- How requirements objects might facilitate traceability from the highest level statement of need to the lowest level of implementation.
- If, and how requirements objects facilitate the engineering of complex systems.
- How Object-Oriented requirements can lead to Object-Oriented Capability Development and provide a way to manage the acquisition of systems of systems.
- Can tools such as CRIP charts really provide early warnings of project problems?

Summary

This paper has provided an overview of an Object-Oriented approach to requirements engineering and management together with some of the anticipated benefits. The properties of the requirement object so far identified contain data and functionality that pertain to the process as well as the product. This integration of information from the three streams of work is different to the current paradigm, which considers the three streams as independent. Further research is called for.

Conclusions

Object-Oriented requirements engineering by virtue of having many of the desirable additions to the current paradigm already built in, forces them to be addressed during the SLC and hence should be able to reduce the contribution of poor requirements engineering and management to project failures. However, Object-Oriented requirements engineering does not stop there. It seems to have several other advantages over the current paradigm.

Recommendations

Perform Object-Oriented systems engineering without the use of "requirements" (Kasser 2002a). The word "requirement" is closely coupled to the functional paradigm focussing on the product needed to provide the solution to the customer's problem.

In the Object-Oriented paradigm, the terminology should change to correspond with the change in focus. Customer's needs should be stated in terms of capability needed. Contractors should develop and provide capability to meet the need. Test and Evaluation should test the capability provided to ensure the need is met, and evaluate the capability provided to determine the boundaries (Kasser 2000a).

"Capability" can be coupled to both the product and the process that produced the product. While requirements and specifications may still be suitable for simple systems, complex systems tend to provide capability. So perhaps in the future a better term for the process of the engineering of complex systems would be Object-Oriented Capability Development.

References

Alexander, I. F., Stevens, S., *Writing Better Requirements*, Addison-Wesley, 2002.

Carson, R.S., "Keeping the Focus During Requirements Analysis", *Proceedings of the 11th International Symposium of the International Council on Systems Engineering (INCOSE)*, Melbourne, Australia, 2001.

Cook S. C., Kasser J.E., Asenstorfer J., "A Frame-Based Approach to Requirements Engineering", *11th International Symposium of the INCOSE*, Melbourne, Australia, 2001.

Dorfman M., Thayer R.H. *System and Software Requirements Engineering*, IEEE Computer Society Press, 1990.

Hooks, I., "Writing Good Requirements", *Proceedings of the 3rd NCOSE International Symposium, 1993,* available at http://www.incose.org/rwg /writing.html, last accessed November 1, 2001.

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

Jacobs, S., "Introducing Measurable Quality Requirements: A Case Study", *IEEE International Symposium on Requirements Engineering*, Limerick, Ireland, 1999.

Hopkins, F.W., Rhoads, R.P. "Object Oriented Systems Engineering – An Approach", *Proceedings of the 8th International INCOSE Symposium*, 1998

Kasser, J.E., *Applying Total Quality Management to Systems Engineering*, Artech House, 1995.

Kasser, J.E., "What Do You Mean, You Can't Tell Me How Much of My Project Has Been Completed?", *Proceedings of the 7th Annual International Symposium of the INCOSE*, Los Angeles, CA. 1997.

Kasser, J.E., "A Framework for Requirements Engineering in a Digital Integrated Environment (FREDIE)", *Proceedings of the Systems Engineering, Test and Evaluation Conference (SETE 2000),* Brisbane, Australia, 2000.

Kasser J.E., (2000a), "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", *SETE 2000*, Brisbane, Australia, 2000.

Kasser J.E., "A Prototype Tool for Improving the Wording of Requirements", *Proceedings of the 12th International INCOSE Symposium*, 2002.

Kasser J.E., (2002a), "Does Object-Oriented System Engineering Eliminate the Need for Requirements?", *Proceedings of the 12th International INCOSE Symposium*, 2002.

Kasser J.E., (2000b), "The Cataract Methodology for Systems and Software Acquisition, *SETE 2002,* Sydney Australia, October 2002.

Kasser J.E., Cook S.C., "Using a Rapid Incremental Solution Construction Approach to Maximise the Completeness and Correctness of a Set of Requirements for a System" *Proceedings of the 13th International INCOSE Symposium*, 2003.

Kasser J.E. and Cook S.C. "The Communications Requirements Evaluation & Assessment Prototype (CREAP)", *Proceedings of the 12th INCOSE,* Las Vegas, NV, 2002.

Kasser J.E., Cook S.C., Scott W, Clothier J., Chen P., "Introducing a Next Generation Computer Enhanced Systems Engineering Tool: The Operations Concept Harbinger", *SETE 2002,* Sydney Australia, 2002.

Kasser, J.E., Schermerhorn R., "Determining Metrics for Systems Engineering", *Proceedings of the 4th International Symposium of the NCOSE*, San Jose, CA., 1994.

Kasser J.E., Tran X-L, Matisons S., "Prototype Educational Tools for Systems and Software (PETS) Engineering", *Proceedings of the 14th Annual Conference for Australian Engineering Education*, Melbourne, 2003.

Kotonya G. and Sommerville I., *Requirements Engineering: Processes and Techniques*, Wiley, 1998.

Lykins, H., Friedenthal, S., Meilich, A., "Adapting UML for an Object Oriented Systems Engineering Method (OOSEM)", *Proceedings of the 10th International INCOSE Symposium*, 2000.

Meilich, A., Rickels, M., "An Application of Object Oriented Systems Engineering (OOSE) To an Army Command and Control System: A New Approach to Integration of System and Software Requirements and Design", *Proceedings of the 9th International INCOSE Symposium*, 1999.

Schach S., *Object-Oriented and Classical Software Engineering*, p 294, McGraw Hill, 2002.

Standish (1995), *Chaos*, The Standish Group, http://www.standishgroup.com/chaos.html, last accessed March 19, 1998.

ST DADS (1992). Requirements Analysis Document (FAC STR-22), Rev. C, August 1992, as modified by the following CCR's:- 139, 146, 147C, 150 and 151B.

Von Knethen A., Paech B., Kiedaisch F., Houdek F., "Systematic Requirements Recycling through Abstraction and Traceability", *Proceedings IEEE Joint International Conference on Requirements Engineering*, 2002.

Voyages (1996), "Unfinished Voyages, A follow up to the CHAOS Report", The Standish Group, http://www.pm2go.com/sample_research/unfinished_voyages_1.asp, last accessed January 21, 2002.