Enhancing the Understanding of Customer Requirements in Business Software Engineering

W. Al-Karaghouli, S. AlShawi, M. Elstob
University of Westminster, Brunel University, Brunel University, UK

A major cause of the failure of information technology system (ITS) projects is the problem of fully understanding customer requirements. In this paper we introduce the idea of a Knowledge Requirement System (KRS) as an approach to overcoming the customer requirements problem. This approach combines in a novel way techniques from operational research (OR) and soft systems methodology (SSM). In particular, it is shown how Venn Diagrams may be used to clarify and establish the relationships and gaps existing between the customer's and the software engineer's knowledge, understandings, and expectations.

Keywords: information technology systems (ITS), customer requirements, specifications, Venn diagrams, knowledge requirement system (KRS), brainstorm, operational research (OR), rich picture, soft systems methodology (SSM).

1. Introduction

In many large and complex information technology systems projects (ITS), the need for clear understanding of customer requirements has often been underestimated, and this has led to an undermining of the success of vital and expensive projects. Just considering the UK, we have for example: the London Stock Exchange automated trading system, Taurus, which after great expenditure had to be withdrawn before ever going live [12]; and [6]; The London Ambulance Service computerised despatch system which failed disastrously forcing the Service to revert to the old system [23]; and more recently the new UK air traffic control centre at Swanwick, which is still not operational three years after its planned completion and is very greatly over budget, [10] and [14].

The problem of failed software is not a characteristic of the UK software industry alone. Early international cases of software failures have not been reported in the past, unless one searches very hard. It is a global problem, for example, Sauer, [26] extensively discusses the famous case of the Australian Public Service Mandata system, which started in the 50s and continued throughout the 70s, eventually being terminated in April 1981 with full cost not revealed. In the United States, where the software industry is flourishing in Silicon Valley, many large-scale systems have ended in disasters, as discussed by Laudon and Laudon [19] and Sloane [28].

A number of authors, including Schmitt and Kozar [27]; Rudelius, Dickson, and Hartley [25]; Wise and Debons [31]; Sommerville [29]; Sauer [26]; Tiller [30]: and Flowers [12] draw attention to many factors that lead to software failures. Several reasons exist for such failures (figure 1), for example cost and time overruns due to poor initial specification, obsolescence due to rapidly changing work environment

[24], and inadequate requirements analysis and problem definition. Macaulay [21] and Gleick [13] argue that some ITS failures are the result of mutual misunderstanding between the customer and the software engineers about vital aspects of the project's requirements. It is this area which represents the focus of our research.

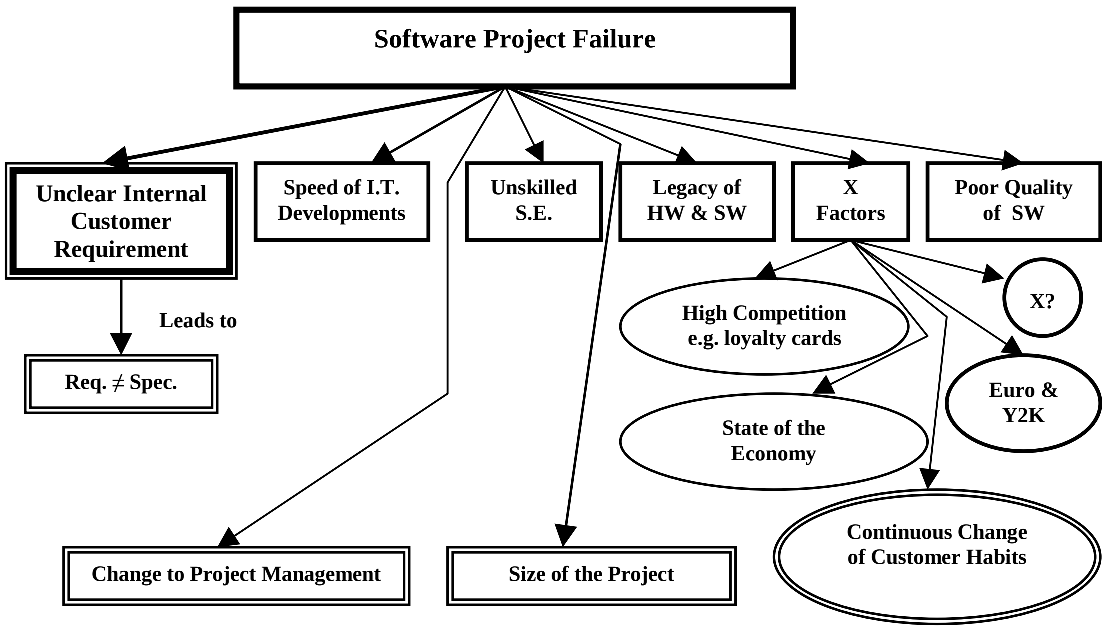

Software Project Failure
Unclear Internal Customer Requirement
Speed of I.T. Developments
Unskilled S.E.
Legacy of HW & SW
X Factors
Poor Quality of SW
Leads to
Req. ≠ Spec.
High Competition e.g. loyalty cards
X?
Euro & Y2K
State of the Economy
Continuous Change of Customer Habits
Change to Project Management
Size of the Project

Figure 1 Factors Contributing to the High Rate of Software Project Failures

The total quality management (TQM) notion of prevention rather than correction can be applied successfully in software engineering, just as it has been effectively applied in many areas, see for example, Al-Karaghouli [2]. We believe that getting the customer "requirement" right first time during the initial stage of the life cycle (Definition of Requirements), rather than at a later stage, will save both the customer and the software engineer (s.e.) time and money, see Ishikawa [15], and Crosby [11].

Thus, intensive and continuos communications between the customer and the software engineer is extremely important to establish a clear understanding of the business need which the proposed system must support in order to get things right first time Juran [16], but in most of the cases this is not happening in the real-world as it has been indicated by Kelly [17] who quotes the software guru "Ivar Jacobson saying:

" Two generations of developers have been lost to bad habits. They do the coding and then debug. They should get it right from the beginning".

Many organisations never had the chance to talk to one another about their new ITS were suddenly they find themselves integrated by the so called efficiency system. In fact, most of the ITS which built in this way had almost worked against the development of such a culture, the reader is referred to Cavell [7], and List [20] for recent work on the implementation of such ITS in the retail sector.

Land [18], and Abdel-Hamid & Madnick [1] also highlight the importance of learning from failures, and of the vital need of the software engineers to clearly understand the customer's requirements. We take a slightly broader view in that we see the problem not only being that the software engineers often fail to understand the customer's business and needs, but that the customers in turn often do not sufficiently appreciate the realities of software development, or what the software people are offering. Our aim is to develop techniques and tools to help overcome this set of problems. On the one hand we are developing methods to help identify and make mutually apparent the gaps that exist between the understanding that each side in the project has, and on the other hand we have techniques aimed at facilitating and accelerating the generation of understanding to close these gaps, see Al-Karaghouli [3] & [4].

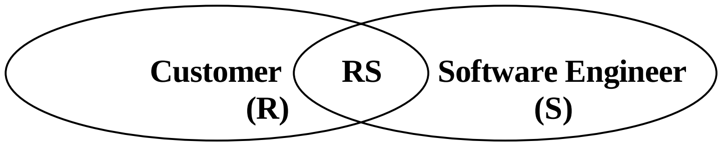
Building and developing future business software should be viewed a sort of self-service; the actual development of the product should not be framed out to the customer. What we are preaching (if the word "preach" is the correct terminology) here is how the world has changed from 50 years ago when the 1st computer system has been built, as international boarders between countries become blurred so the difference between business (customer) and the software engineer (developer) is getting blurred too.

2. Identifying the Understanding Gap (UG)

Venn Diagrams have been used successfully for some time in the management science [5], as well as, of course, in their traditional areas of logic and computing. In our work we have found that they are highly effective as a pictorial technique for illustrating the gaps in understanding that exist at the requirements stage. They are extremely easy to understand and can be manipulated by both sides to make particular points. For example, by re-negotiating the overlaps it is easy to indicate how good or bad current agreements are on particular matters. The technique can be made more precise and quantitative by, for example, creating a matching score of specific terms noted in attribute lists drawn up by the two sides.

The following diagrams and discussion give some indication of the ways in which we have found Venn diagrams can be particularly helpful.

GENERAL CASE:



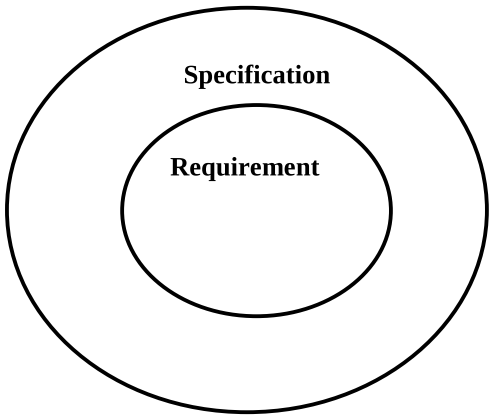
Requirement (customer) $\neq$ Specification (software engineer)

What the above diagram clearly illustrates is that the customer and the software engineer have different perceptions and understandings of what the system is to deliver and what it will be like. Each circle is supposed to indicate all the various features, factors, values, etc. understood and expected of the project by each side.

At the initial stage of discussion about the project, no attempt is made to specify the detailed content of each circle, although later it is very useful to draw up specific lists. At this early stage, it is recommended that each side draw their own overlapping Venn diagram to indicate what they believe the level of mutual understanding currently is. These two diagrams then provide a clear starting point for further discussion with attention being focused on why there is disagreement between each side's view, and on the nature of the current mismatch between requirements and specification. This discussion should then lead to the development of specific further activities to obtain better understanding, from each side, so as to increase the overlap and agreement. A scenario approach is useful to enrich the communications between both parties [21], [22], & [24]. Because of the simplicity and clarity of the Venn diagram, which is especially helpful to the customer, it is recommended that the above process is repeated frequently throughout the requirements definition stage of the project's life cycle.

Venn diagrams, however have many additional uses in the requirements definition process. For example, they can be used by the software engineer to illustrate various undesirable situations that may arise and which both sides need to avoid. Two such cases are shown in the diagrams below.

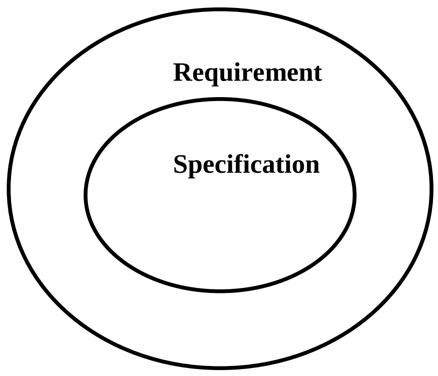
CASE 1:



Specification greatly exceeds requirement.

This may lead to the customer greatly over spending on the project for facilities and equipment that are not needed.

CASE 2:



Specification meets only part of the requirement, but does not include any unneeded facilities and equipment.

This is a less worse situation than Case 1, but it is still unsatisfactory since there are important customer requirements that have not been met.

A more complex situation that often arises and is sometimes not well understood by customers is illustrated in figure 2 below.

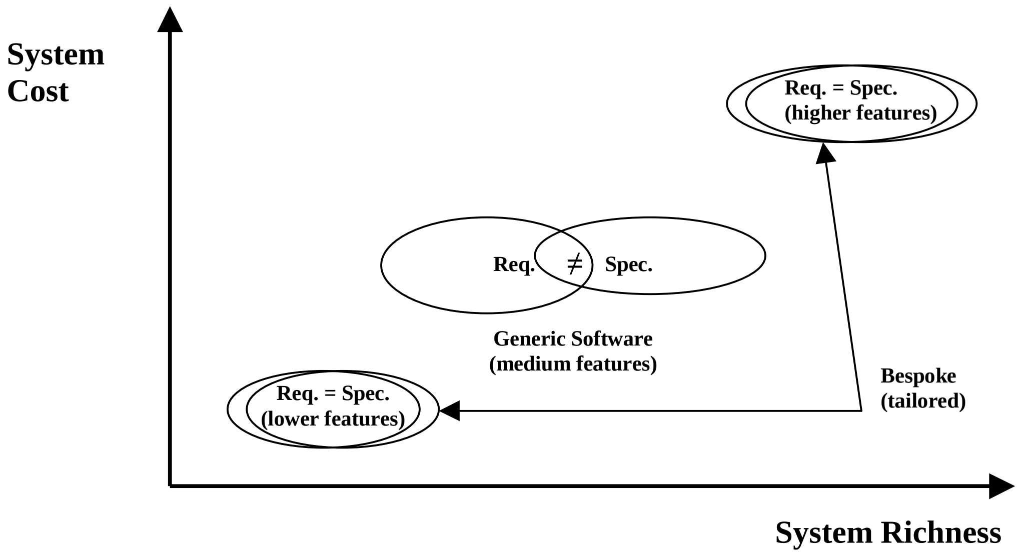

System Cost
Req. = Spec. (higher features)
Req. ≠ Spec.
Generic Software (medium features)
Bespoke (tailored)
Req. = Spec. (lower features)
System Richness

Figure 2 The Conceptual Paradigm of System and Customer Requirements Correlation

This shows the relationship between system costs and the breadth and depth of the features of the proposed system. It is possible to have a low cost system meeting basic requirements with little possibility of adaptation and expansion, see 'lower features' Venn diagram. Alternatively, a 'higher features' bespoke system can be produced at much greater cost, however with possibly lower system lifetime costs and certainly much greater flexibility for expansion and development, in both cases the "lower features" and the "higher features", we can identify a strong correlation between system costs and the features of the proposed systems as a measure of customer requirement being satisfied by the ITS. Finally, there is the common generic software problem, as shown by the 'medium features' Venn diagram. In such a case a fairly fully featured packaged system is available that seems to offer excellent value for money. However, such packages often leave many important customer requirements unsatisfied.

3 Knowledge Requirements System (KRS)

Clearly, the Venn diagram technique, although very useful, is limited in its ability to stimulate deeper understandings and reveal major areas of disagreement and mismatch of goals, values, and expectations. Our overall approach, which we refer to as the KRS approach, as illustrated in figure 3, is aimed at generating the knowledge and understanding needed in both the customer and the software engineers to enable the development of a sound, agreed, and fully effective project requirements definition. The key principle that drives our approach is the belief that both sides must work together to generate knowledge and understanding of one another's worlds and that both have a lot of learning to do. We reject the view that the customer already knows fully exactly what they want of the system, and we also reject the view that the software engineer knows fully about the customer's business and what is needed of the system. We believe that the only way to get good requirements definition is through partnership and mutual understanding.

Customers generally either get too enthusiastic about technology and hopelessly over-estimate the technology's capacity to change the world, or they never hang around long enough to get "up to speed" with technology. ITS engineers must adapt to a world in which many business models are different from those they know. There are several messages to the ITS engineers, one would be to look at the engineering of the product (software) without boundaries. Developing large ITS is complex task, we need everybody to work as engineers (the customer plus the software engineer), it has to be done this way in order to make things which are complex work.

KRS consists of four major levels and a total of ten processes within these levels, one tool namely Brainstorming is used across all the four different levels, that contain the processes. In addition, two extra tools are used in the second level to achieve the needed and already specified quality requirements.

KRS in contrast to normal technical systems, such as Database Management System (DBMS), KRS is a sociotechnical approach deals mainly with social (human) interactions between the customer and the software engineer. After all the process of customer requirements elicitation involves human-human communications (e.g. various interview techniques) with the aim of setting up the proper technical specifications of the proposed systems, i.e. getting specific customer requirements right first time is a sociotechnical issue which KRS approach provides a feasible solution.

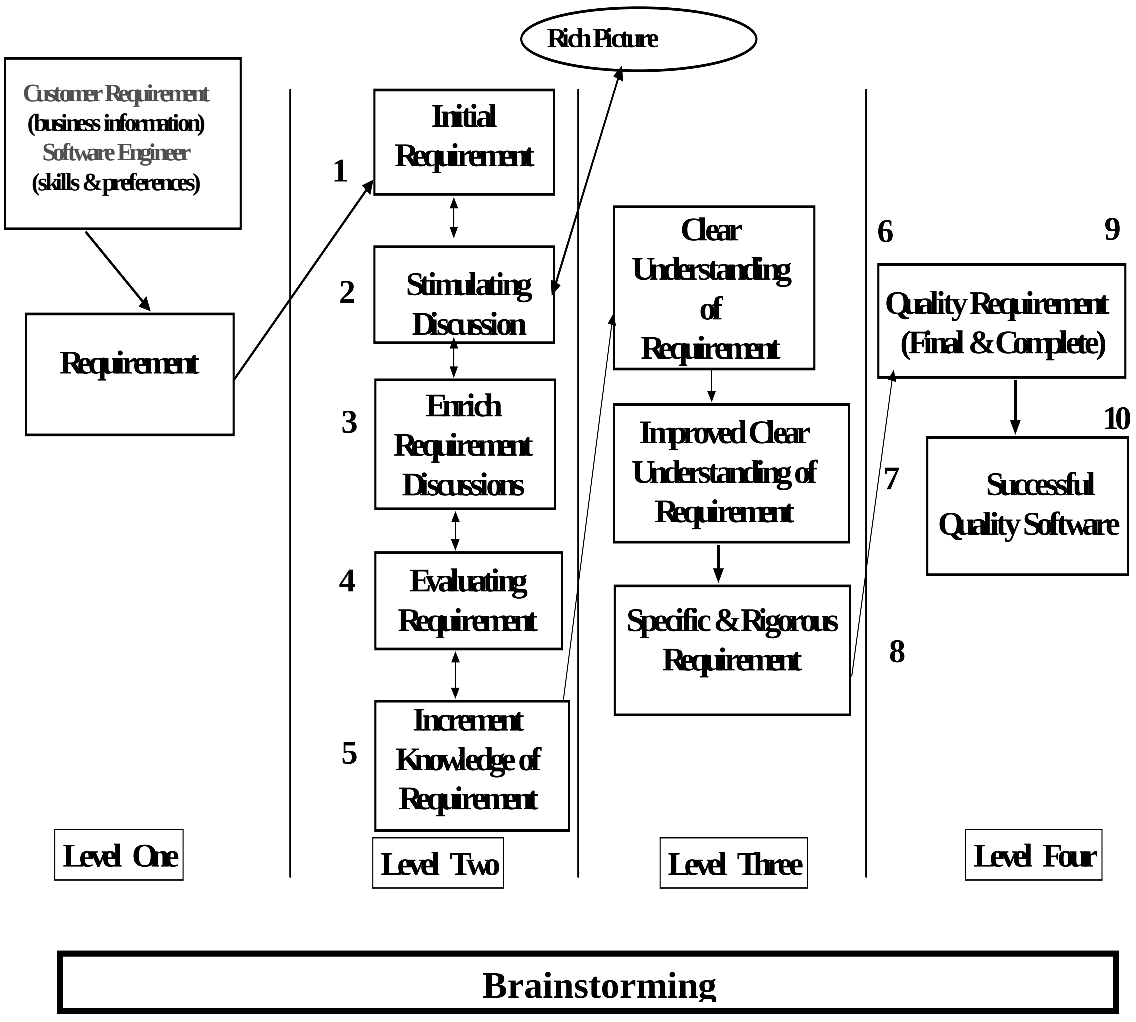
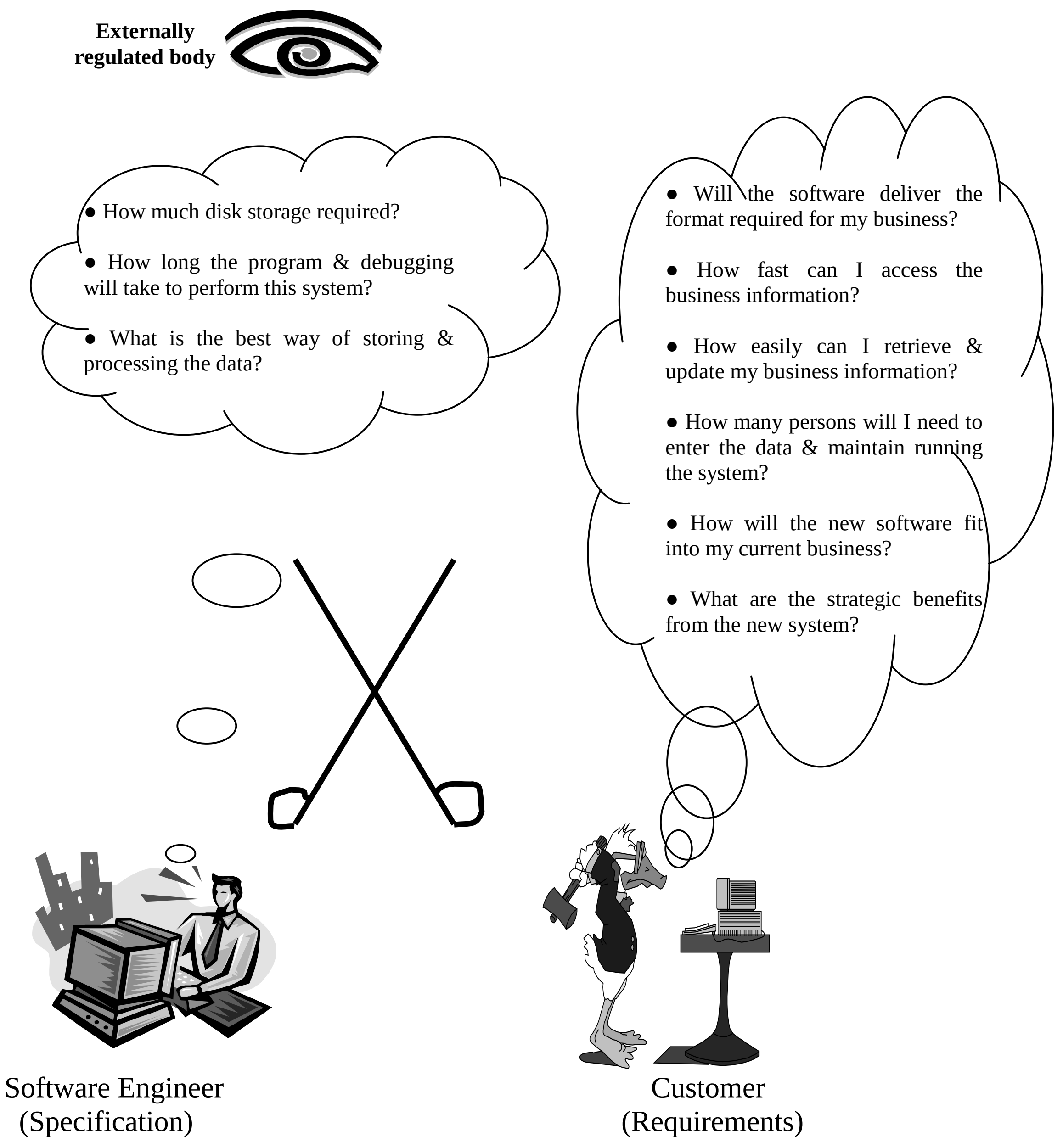

Rich Picture
Customer Requirement
(business information)
Software Engineer
(skills & preferences)
Requirement
1
Initial Requirement
2
Stimulating Discussion
3
Enrich Requirement Discussions
4
Evaluating Requirement
5
Increment Knowledge of Requirement
Clear Understanding of Requirement
Improved Clear Understanding of Requirement
Specific & Rigorous Requirement
6
7
8
9
Quality Requirement (Final & Complete)
10
Successful Quality Software
Level One
Level Two
Level Three
Level Four
Brainstorming

Figure 3 Structure of the Knowledge Requirement System (KBS)

KRS uses several techniques at present, and we are also exploring the use of further methods. The purpose of all of these tools is to generate, in the various parties involved in a project, the knowledge and understanding needed to create an effective requirements definition. One of the techniques that we have used successfully is the well-known Soft Systems Methodology (SSM) and in particular the Rich Picture technique [8], as illustrated in figure 4. SSM focuses, and concerns with the entire project-changing environment, it resolves conflicts which are not addressed by most formal analysis approaches.


Externally regulated body
● How much disk storage required?
● How long the program & debugging will take to perform this system?
● What is the best way of storing & processing the data?
● Will the software deliver the format required for my business?
● How fast can I access the business information?
● How easily can I retrieve & update my business information?
● How many persons will I need to enter the data & maintain running the system?
● How will the new software fit into my current business?
● What are the strategic benefits from the new system?
Software Engineer (Specification)
Customer (Requirements)

Key to the above rich picture:

Externally interested party:	👁
Possible conflict:	✕
Major concern:	💭

Figure 4 A Simple Rich Picture Illustrating a Possible Gap Between the Software Engineer and the Customer

The Rich Picture methodology is much more extensive than the above figure indicates and in KRS we are employing the full range of its techniques.

4 Conclusion

In this paper we have introduced the use of Venn diagrams as a convenient and powerful pictorial aid to developing shared understanding and knowledge between the customer and the software engineer in ITS projects.

This technique is part of a larger programme of research, called the Knowledge Requirement System approach, which is aimed at producing a range of tools and methods to elicit, and facilitate the effective execution of the requirements definition stage of projects. The work is ongoing and at present includes methods drawn from Soft Systems Methodology, Brainstorming, and diagrammatic modelling.

The authors wish to thank the University of Westminster for their support; MBA; DMS students at Westminster Business School; and large UK retail organisations for their participation in this research.

References:

[1] T.K. Abdel-Hamid, and S.E. Madnick, 'The Elusive Silver Lining: How we Fail to Learn from Software Development Failures', Sloan Management Review, 39, 1990 pp.39-48.

[2] W. Al-Karaghouli, 'Total Quality the Way Forward: Implementation, Achievements and Success', Total Quality Conference, The Polytechnic of Central London, May, England, 1991.

[3] W. Al-Karaghouli, '£ Millions Lost on Software Engineering: Myth or Reality?', Unpublished Seminar Notes- Research Seminar Series, Westminster Business School, University of Westminster, June, England, 1997.

[4] W. Al-Karaghouli, 'Failures of Software Engineering in Retail: The Story So Far', Unpublished Seminar Notes- Research Seminar Series, Westminster Business School, University of Westminster, February, England, 1998.

[5] D.R. Anderson, D.J. Sweeney, and T. A. Williams, 'Quantitative Methods for Business', 6th Edition, West Publishing Company, New York, 1995.

[6] P. Beynon-Davies, 'Information Systems: Failures and How to Avoid Them', Financial Times Management Briefings, Pitman, England, 1996.

[7] S. Cavell, 'Salespeople Buck the System: Survey Finds Software Fails to Take Account of Culture', Computing 25th February, 1999, pp.16.

[8] P. Checkland, and J. Scholes, 'Soft Systems Methodology in Action', Wiley & Sons, Chichester, England, 1997.

[9] Computing, 'Stock Exchange Kills Projects to Focus on Taurus', UK, 2nd, November, 1989, pp.1.

[10] Computing, 'Swanwick Air Traffic Centre Set for More Delays', UK, 22nd October, 1998, pp.23.

[11] P. Crosby, 'Quality Improvement Through Defect Prevention', Philip Crosby Associates, Inc., New York, 1985.

[12] S. Flower, 'Software Failure: Management Failure', John Wiley, England, 1996.

[13] J. Gleick, 'Chaos: Making a New Science', Penguin Books, Harmondsworth, 1998.

[14] L. Hatton, ‘Swanwick Bug Fixes not Good Enough’, UK, Computer Weekly 7th January, 1999, pp.11.
[15] K. Ishikawa, ‘What is Total Quality Control? -The Japanese Way’, Translated by D.J. Lu. Prentice Hall, New York, 1985.
[16] J.M. Juran, ‘Juran on Leadership for Quality: An Executive Handbook’, 4th ed. McGraw-Hill Book Company, USA, 1989.
[17] L. Kelly, ‘Developers Bark up Wrong Tree’, Computing 4th March, 1999, pp.14.
[18] F. Land, ‘Adapting to Changing User Requirements’, Information and Management, 5, 1982, pp.59-75.
[19] K.C. Laudon, and J.P. Laudon, ‘Management Information Systems’, Prentice Hall, New Jersey, 1996.
[20] B. List, ‘Managing Software Development and Maintenance’, OR/MS Today, February, 1999, pp.69.
[21] L. A. Macaulay, ‘Requirements Engineering’, Springer-Verlag, London, 1996.
[22] N.A.M. Maiden, S. Minocha, M. Ryan, K. Hutchings & K. Manning, ‘A Co-operative Scenario-Based Approach to Acquisition and Validation of System Requirements: How Exceptions can Help!’, Proceedings Workshop on Errors in Systems Development, University of Glasgow, Scotland March 20-22, 1997.
[23] D. Page, P. Williams, and D. Boyd, ‘Report of the Inquiry into the London Ambulance Service’, Southwest Thames Regional Authority- February, England, 1993.
[24] Rolland C., C. Souveyet, and C.B. Achour, ‘Guiding Goal Modelling Using Scenarios’, IEEE Transactions on Software Engineering: Special Issue on Scenarios Management, Vol.24, No.12, December 1998.
[25] W. Rudelius, G.W. Dickson, and S.W. Hartley, ‘The Little Model That Couldn’t: How a Decision Support System for Retail Buyers Found Limbo, Systems, Objectives, Solutions’, 1982, Chapter 2, pp.15-124.
[26] C. Sauer, ‘Why Information Systems Fail: A Case Study Approach’, Alfred Waller, Henley-on-Thames, UK, 1993.
[27] J.W. Schmitt, and K.A. Kozar, ‘Management’s Role in Information Systems Development Failures: A Case Study’, MIS Quarterly, 2nd June, Vol.22, 1978, pp.7-16.
[28] S.B. Sloane, The Use of Artificial Intelligence by The United States Navy: Case Study of a Failure, AI Magazine, Spring, (1991) 80-92.
[29] I. Sommerville, Software Engineering. Addison-Wesley, 1992, USA.
[30] A. Tillier, Smartcards Fail to Impress French GPs, Computing 18th February, (1999) 16.
[31] J.A Wise, and A. Debons, Information Systems: Failure Analysis, NATO Advance Science Institute, Series F: Computer and Systems Sciences, 32 (1987) Springer-Verlag. Berlin. Heidelberg.