Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin

Paul Grünbacher
Johannes Kepler University Linz
Systems Engineering and Automation
Altenbergerstr. 69, 4040 Linz, Austria
pg@sea.uni-linz.ac.at

Robert O. Briggs
GROUPSYSTEMS.COM
1430 E. Fort Lowell Rd. #301, Tucson, AZ
bbriggs@groupsystems.com

Abstract

Defects in the requirements definition process often lead to costly project failures. One eminent problem is that it can be difficult to take deliberate advantage of important tacit knowledge of success-critical stakeholders. People know more that they can ever tell. Implicit stakeholder goals, hidden assumptions, unshared expectations often result in severe problems in the later stages of software development. In this paper, we will present a set of collaborative techniques that support a team of success-critical stakeholders in surfacing tacit knowledge during systems development projects. We will discuss these techniques in the context of the EasyWinWin requirements negotiation methodology and illustrate our approach with examples from real-world negotiations.

1. Introduction

Developing software-intensive systems is a high-risk undertaking for technology companies and professional services organizations that derive their revenue from these projects. Failed or delayed projects directly impact the bottom line, and developing a solution with the wrong features negatively affects revenues and market position.

In 1995 the Standish Group [23] studied more than 8000 software projects in 352 companies and found that

- 30% of all projects are cancelled before completion,
- 70% of the remainder fail to deliver the expected features,
- the average project overruns its budget by 189% and overshoots its schedule by 222%.

The study reported that more than half of these projects were compromised from the outset by shortcomings in the requirements definition process (Figure 1). Improvements in the requirements definition process could therefore substantially reduce many of the risks of these projects.

Recent advances in requirements engineering research [20] has addressed many of the problems shown in the Standish Group study:

- There has been a move away from focusing on requirements modeling without understanding the organizational and social context [12,15].
- There is an increased emphasis on approaches modeling the system environment together with the system by focusing on stakeholder goals, scenarios of usage, etc.
- The goal of specifying consistent and complete requirements is seen as futile, and there is an increased focus on negotiation techniques, for analyzing and resolving conflicting requirements.

Requirements emerge in a highly collaborative and social process that involves many stakeholders: the users

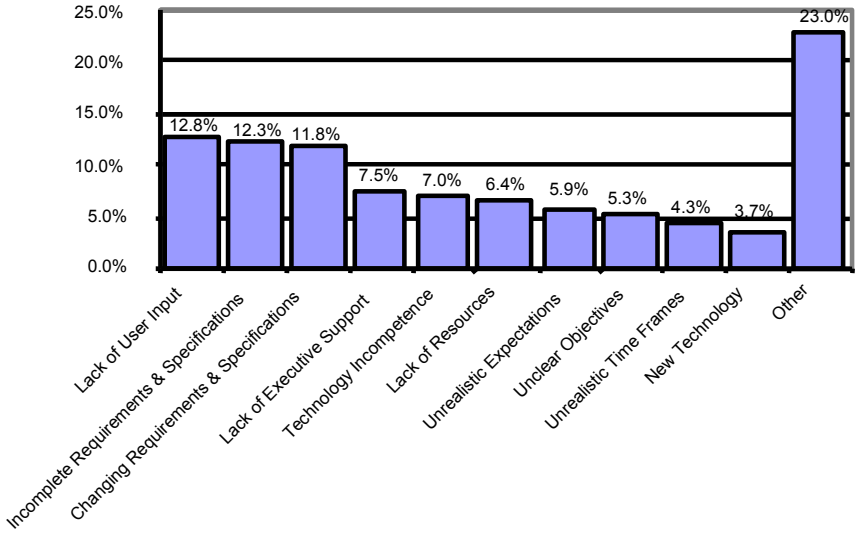

Figure 1. Factors that cause software projects to be cancelled [23]

and the customers, the domain experts and the developers, sales, marketing, and management, to name but a few. Requirements must be negotiated because conflict is inherent in software engineering projects. Early in the project stakeholders contribute incomplete, vague, and often inconsistent statements and ideas about their objectives, assumptions, and expectations. As they work together to negotiate their requirements, they give the project shape, and their merged visions emerge into a system that all stakeholders can accept. If, on the other hand, the stakeholders do not negotiate together, there is little chance the resulting system will accommodate their needs, and the project will often fail. Negotiation is, therefore, essential to achieve mutually satisfactory agreements [2].

One common cause of poor requirements is that critical knowledge of stakeholders remains often hidden and unshared in the course of a negotiation. This may have severe negative consequences for the outcome of the requirements negotiation process:

- If stakeholders are not able to articulate their objectives thoroughly, the emerging requirements will show an incomplete picture.
- The inability of a team to surface tacit knowledge also decreases the capability of a team to identify conflicts among requirements. Conflicts remaining hidden until late in the project often derail a system development effort or result in costly repairs.

Collaborative techniques surfacing tacit knowledge in a requirements negotiation have the potential to reduce major risks. We have been developed a set of techniques in the EasyWinWin project, a groupware-supported methodology for collaborative requirements negotiation [10, 11, 5].

This paper is organized as follows: Section 2 discusses dimensions of conflict in requirements. Section 3 examines problems related to surfacing tacit knowledge not only in the context of identifying requirements, but also for finding and resolving conflicts among requirements. Section 4 introduces the EasyWinWin methodology and presents collaborative techniques that assist in surfacing tacit knowledge and help to reveal a broader set of stakeholder objectives, expectations, and domain knowledge. Conclusions round out the paper in Section 5.

2. Requirements conflicts

Conflict is an inevitable part of both requirements definition and system design [8]. Suppressing or overlooking conflicts is risky and might have serious negative effects on the software development process. Understanding requirements conflicts is thus an important strategy to mitigate software development risks. Many authors have emphasized the importance of identifying and analyzing conflicts for the success of system development [3, 6, 16, 17, 21]: Conflicting requirements must be identified and negotiated, relevant alternatives must be made explicit and it must be assured that the "right" decision is made. Taking into account different and potentially conflicting views avoids a narrow frame and helps to build an appropriate picture of a problem.

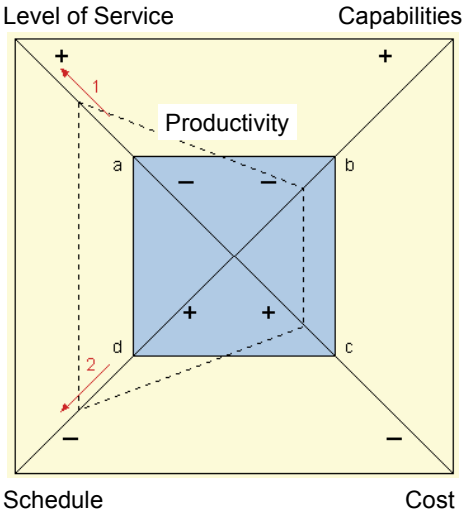

Figure 2. The devil's square

The commonly used devil's square (see Figure 2) illustrates some *dimensions of conflicts* that have to be negotiated. A team of stakeholders has to deal with five immutable dimensions of conflicts in a requirements negotiation process:

- Quantity of capabilities,
- desired level of quality,
- budget,
- schedule,
- and productivity.

In a requirements negotiation, stakeholders represent these dimensions of concern. Depending on the constraints of a project these dimensions are either variant or invariant. The outcome of a dimension can only be changed if other dimensions are adjusted accordingly. Two examples:

- Given that the productivity/capacity of the development team is invariant (=the gray area in Figure 2), additional capabilities can only be delivered by increasing the budget and/or schedule or reducing the desired level of quality.
- If quality, budget and schedule are invariant, the only way to increase the quantity of deliverables is to increase the productivity/capacity of the development team.

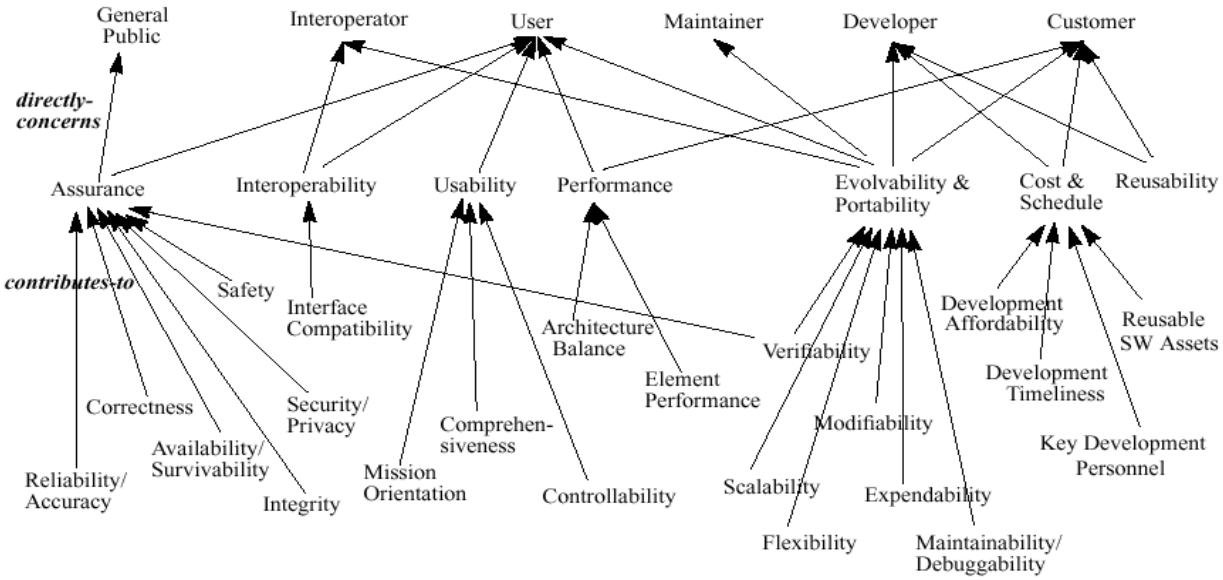

Figure 3. Typical stakeholder concerns in requirements negotiation [Boehm-96]

The *different roles* of stakeholders inevitable lead to conflicts. For example, customers and users are rarely the same. Users may want context-sensitive help to increase usability of the system, whereas customers try to reduce the budget by implementing a simple lookup help system. Figure 2 shows typical stakeholders and their concerns.

Besides role related conflicts *individual interests*, goals and expectations of stakeholders may clash. For example, a software engineer's personal career goals might result in the objective to use cutting-edge technology for development. This might clash with the organization's win condition to adopt proven, reliable methodologies and technologies.

While there is a general consensus about the importance of understanding and resolving conflicts among requirements, this task is very difficult to achieve. Surfacing conflicts is often hampered because so much of the knowledge required to find these conflicts is tacit; i.e. people know something, without being able to articulate it.

3. Tacit stakeholder knowledge

People know more than they can ever tell. Philosophers call this situation the problem of tacit knowledge, i.e. the phenomenon that people may know how to do something, without being able to articulate how they do it. Tacit knowledge is connected with perceptions, ideas, and experience. Generally, it is difficult to transfer and acquired through practice and experience [18].

When this tacit knowledge concerns a proposed software system, we are often at a loss how to bring forth this knowledge. Experience shows that simply asking stakeholders what they want often works poorly. They cannot accurately describe their needs. If tacit stakeholder knowledge remains hidden the following problems occur:

- Incomplete requirements because “obvious” ideas are not captured
- Reduced ability to identify conflicts because not all project-relevant knowledge is explicitly available
- Conflicting interpretations due to terminology differences
- Hidden stakeholder expectations and assumptions not available explicitly

The challenging issue is if elements of our tacit knowledge are inaccessible in principle [25], or just hard to verbalize, and remain either consciously or unconsciously suppressed [24]?

4. Surfacing tacit knowledge in EasyWinWin

We have elaborated a sequence of collaborative techniques assisting in bringing forth this tacit knowledge. The goal of our approach is to access the combined expertise and knowledge of stakeholders as early as possible in a way that optimizes limited stakeholder availability and that doesn’t waste cognitive effort.

We have developed EasyWinWin, a collaborative requirements negotiation methodology. EasyWinWin is based on the WinWin negotiation model:

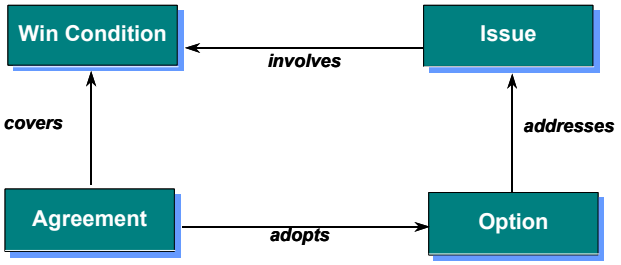

Figure 4. WinWin negotiation model

EasyWinWin combines the WinWin Spiral Model of Software Engineering [1, 2, 4] with collaborative knowledge techniques and automation of a Group Support System (GSS)[1]:

The WinWin negotiation model defines a set of artifacts guiding stakeholders through a negotiation process (see Figure 4). Objectives of stakeholders are captured as *win conditions*. Conflicts among win conditions are recorded as *issues*. *Options* are proposed to reconcile Issues. *Agreements* are developed out of win conditions and out of options by taking into account the preceding decision process and rationale.

A Group Support System (GSS) is a suite of software tools that can be used to create, sustain, and change patterns of group interaction in repeatable, predictable ways [19]. Each GSS tool can be used to create specific group dynamics. For example, an electronic brainstorming tool might be used to cause a group to diverge from comfortable patterns of thought, seeking farther and farther afield for new ideas. A categorizing tool, on the other hand, might be used to cause a group to converge quickly on just the key issues that are worthy of further attention. A group outlining tool might let a group organize complex ideas into an understandable structure, while an electronic polling tool could be used to provoke discussions that uncover unchallenged assumptions and reveal unshared information.

Extensive research in the lab and in the field reveals that, under certain circumstances, teams can use GSS to become substantially more productive than would otherwise be possible [9]. Field studies regularly report that teams using GSS can cut their labor hours for a project by as much as 50%, and can cut the calendar days for their projects by 70-90% [22, 7]. (See [9] for an exhaustive compendium of GSS field research).

Because GSS can be used to create repeatable patterns of group interaction, it can be used to create collaborative methodologies that produce deliverables of consistent quality and detail.

[1]This study was conducted with GroupSystems software developed at the University of Arizona and commercialized by GroupSystems.com.

EasyWinWin is based on the WinWin requirements negotiation model and helps a team of stakeholders to gain a better and more thorough understanding of the problem and supports co-operative learning about other's viewpoints. Different stakeholders – users, customers, managers, domain experts, and developers – come to the project with different expectations and interests. Developing requirements is a learning process: Developers learn more about the customer's and user's world, while customers and users learn more about what is technically and economically possible. This interactive learning process is a prerequisite for the creation of systems satisfying all people who are involved [14, 13].

Teams use EasyWinWin throughout the development cycle to develop:

- Shared Project Vision
- High-levels Requirements Definition
- Detailed requirements for features, functions, and properties
- Requirements for transitioning the system to the customer and user

The nominal purpose of the EasyWinWin methodology is to create an acceptable set of system requirements [10, 11, 5]. However, seven of the eight steps in the methodology are also designed to elicit and capture unshared tacit knowledge that has a direct bearing on the system requirements. In particular, these steps cause the team to identify and then negotiate resolutions for conflicts in system requirements.

This section summarizes each step of the methodology and describes how tacit knowledge emerges during those steps.

Step 1. Refine and Expand Negotiation Topics. The participants view a shared outline containing a taxonomy of system requirements – an outline of categories for the many ways to win during system development. Participants review this outline and make suggestions on how to tailor it to the specifics of their project. The review of the outline serves several purposes:

- It reminds the participants to consider many concepts they might otherwise overlook. For example, a user might not think to state win conditions about system performance.
- As they prune some parts of the taxonomy and elaborate others, assumptions about and constraints on the project emerge. These are explicitly captured in the GSS for future reference.
- The resulting taxonomy becomes an organizing framework for emergent win conditions and becomes a completeness checklist at any time in the project to test whether more work is required.

This steps elicits tacit knowledge by pointing to stakeholders to a list of negotiation topics and asking what's missing and what doesn't belong to the list. The tacit knowledge of stakeholders triggers the customizing of the negotiation space.

Step 2. Brainstorm Stakeholder Win Conditions. The participants surface as many different possible win conditions as they can in a short period of time. A team of 10 stakeholders typically contributes about 300 ideas in an hour-long brainstorm. Here are three examples of win conditions submitted by stakeholders in a negotiation about a new portal.

What we really need is context-sensitive help. We really need it.

I want to develop this with Enterprise Java Beans.

Corporate logo everywhere.

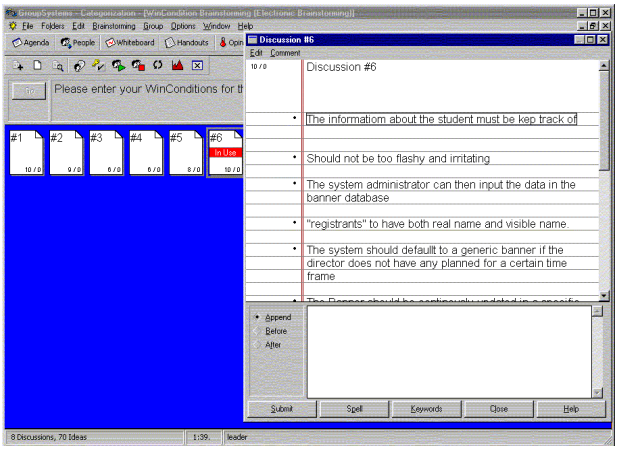

Figure 5. Brainstorm Stakeholder Interests

Because this step is conducted anonymously, people may contribute their strongest and fondest wishes for the system with no fear of retribution from peers or superiors. People may challenge one another's ideas during the brainstorming without creating hurt feelings or incurring social sanctions. The challenges often surface tacit knowledge in the form of unshared expectations and unexamined assumptions.

Because all contributions can be seen by the other participants on their screens, the contributions from one participant remind others of important issues they may not yet have considered. Because the participants contribute simultaneously, nobody needs to wait for a turn to speak; all may contribute the moment an idea occurs.

Because the contributions are anonymous the focus of the group members tends to shift from "My Win" to "Our Win".

Step 3. Converge on Win Conditions. The contents of a brainstorming session tend to be free-ranging, wordy, full of redundancy and irrelevancy, so in this step the team converges on a non-redundant, unambiguous list of win conditions. To do this the group uses an oral conversation supported by two collaborative tools. One tool divides the brainstorming comments among the participants so each sees a different set. This reduces information overload and allows people to work in parallel. The other tool provides them a shared list which all can see on their screens. Drawing from the brainstorming comments on the screen, each participant in turn proposes orally a clear, concise statement of win conditions for the shared list. When all participants have contributed a win condition to the shared list, the system reshuffles the raw brainstorming comments so each person now sees a different set. They review the new set to see if they can identify new win conditions. They continue to swap raw brainstorming comments and pose new win conditions until nobody can find anything new to add to the shared list.

The group discusses each win condition to create a shared understanding of its meaning. Participants may argue about the meaning of any win condition, but they may not object to any win condition at this time. All issues and objections must be reserved for a later step.

During these oral discussions new win conditions that were not part of the original brainstorming session often emerge, and are added to the shared list. Key terms surface during these conversations which may take on special meaning for the project, or which the team may find vague or confusing. These terms are captured to a private list for further processing in the next step.

There are typically about 1/3 to 1/2 as many cleanly stated win conditions as there were brainstorming comments.

Although tacit knowledge is difficult to articulate, as people consider and discuss win conditions posed by others, they gain insight into the tacit knowledge that gave rise to the win condition.

Step 4. Define a Glossary of Key Terms. In any system development project there are key terms that become insider jargon for project members. This step captures the tacit knowledge about the project-specific meaning of these terms. All the key terms derived from the brainstorming session are posted to a shared list. The team breaks into pairs and each pair works out a definition of several key terms and posts the definitions to the shared list. Then the pairs report their definitions to the group orally, which usually provokes spirited debate. The team negotiates an agreed meaning for each term, and usually finds there are other key terms, which should be added to the list and defined.

The captured definitions are valuable throughout the project, especially as the composition of the team changes

over time. However, there is even more value in the spirited debate. As people negotiate the meanings of words, key project constraints emerge, assumptions surface, and the team frequently identifies new stakeholders who should be included in the requirements process.

Often key terms turn out to be confounded, having many meanings. In one recent case, the term, "affiliate" turned out to have five different meanings. Each of those meanings was given a new, more specific term, and those five terms were added to the list. Other times it turns out that several key terms have the same meaning. The team chooses one label for the construct and deletes all other redundant labels.

This step may be repeated several times throughout the project as the team collects new terms.

Once the terms have been defined the team goes back and restates the win conditions more precisely. This step helps to develop a mutual understanding of language and to eliminate ambiguous statements.

Step 5. Prioritize Win Conditions. The participants rate each win condition along two criteria:

- *Business Importance*: The degree to which the success of the project depends on this win condition being realized.
- *Ease of Realization*: The degree to which a win condition is technologically, socially, politically, and economically feasible.

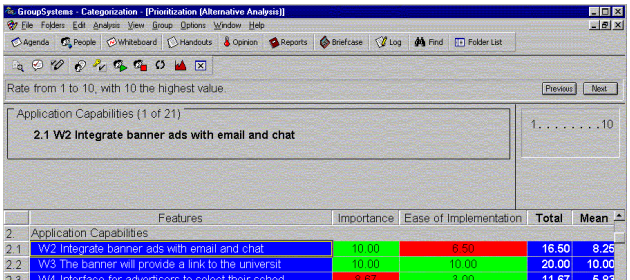

Figure 6. Prioritize Win Conditions

During this assessment, the participants are instructed, "If you don't know, don't vote." Customers and users often decide not to render opinions about the ease of realization. Programmers frequently choose not to rate the business importance of a given win condition. Some people offer no assessment of win conditions in which they have no stake, focusing instead on the ones about which they care.

This is the first step where participants are allowed to register an opinion about the win conditions. The results are a visible artifact of tacit knowledge, approximating unarticulated hunches, hopes and concerns. However, the results are not used to eliminate any win conditions. Rather they are used to provoke a well-structured, very contained exploration in the next step.

Moreover, the step allows the individuals to see how their own opinion compares to that of the group, and this helps them to learn about expectations and perhaps to identify unreasonable expectations of their own.

Step 6. Surface Issues and Constraints. The GSS displays the average assessment of each win condition for each criterion. Items where the group has high consensus display in green. Items where the group has low consensus display in red. A click on a red item displays a graph and a table that reveals the pattern of votes. The group holds a structured conversation to try to explain what reasons might exist for giving an item a high rating, and what reasons might exist for giving an item a low rating. Several kinds of tacit knowledge emerge during these attempts to explain the voting patterns:

- Project constraints – these are captured as Issues associated with a particular win condition.
- Assumptions – these are captured as electronic annotations attached to a win condition. In one example there was a win conditions to forbid the use of CGI-scripts in development. The vote reveals a bi-modal split in the group. It turned out that some stakeholders assumed the system would be managed by an external group that forbade the use of CGI-scripts. Others assumed that the system would be managed internally. By challenging both these assumptions the group was able to identify a new project constraint.
- Unshared information – captured as electronic annotations to a win condition. For example, in one team with which we recently worked 12 stakeholders rated on particular feature as extremely ease to realize. One stakeholder rated the same item as extremely difficult to implement. During the follow-up exploration the one stakeholder shared his insight knowledge and previous experience and suggested the task would be far from trivial. Instead of going with majority rule the avoided what would otherwise have been a nasty pitfall.
- Hidden agendas – captured as new win conditions.

Step 7. The WinWin Tree: Win Conditions, Issues, Options, Agreements. The team posts a shared outline with all the win conditions as main headings. The team makes three passes through this outline. On the first pass each person reads each win condition. If the win condition raises any issue with a stakeholder, the stakeholder may write the issue as a sub-heading to the

win condition. The participants may not discuss the issues aloud at this time.

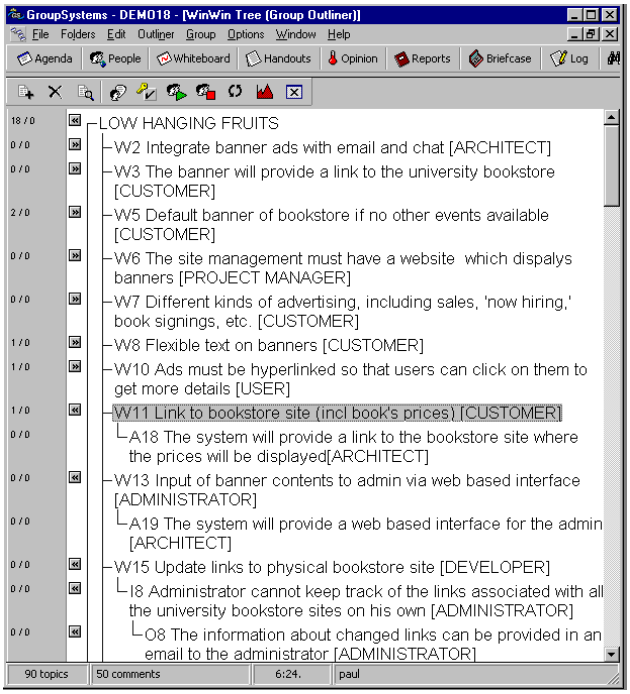

Figure 7. The WinWin Tree

On the next pass, each participant reads each issue. If a participant can think of any option for resolving the issue, the participant may write the option as a sub-heading to the issue.

During the issues section, people notice when someone else's win condition is in conflict with their own tacit knowledge. They can stop and reflect on this without losing the thread of the group's interaction. It's hard to do in an oral process or by handing them a requirements document.

Once the issues and options have been articulated, the group is ready to begin negotiating agreements. There are usually no issues on about 1/3 of the win conditions. After a quick review, these items are declared to be automatic agreements. They become commitments the team must fulfill. Then the group addresses each issue in turn with an old-fashioned oral negotiation.

Sometimes one or more of the options posted with an issue turn out to be the basis for an agreement. Other times the stakeholders engage in protracted discussions of an issue. During that conversation, all manner of useful tacit knowledge emerge – more assumptions and constraints, more key terms, more options, more win conditions. Each of these is captured in its place on the WinWin Tree. Every time a team member proposes an agreement out loud, somebody types it as an option on the tree. As people argue for and against options, someone captures pros-and-cons as electronic annotations to the options. Eventually, the group fashions an agreement with which they can live. They write an agreement as a subheading to the option. When every win condition and every option has an agreement, the state of WinWin Equilibrium has been achieved.

This steps leverages tacit knowledge about issues and options. The group's knowledge about potential risks and problems emerges. An issue may appear not resolvable to the posers of the win condition and that issue. However, another person might identify an easy answer.

Step 8. Organize Negotiation Results. Stakeholders post all their win conditions on a shared list. Next to that they post a set of electronic buckets representing the taxonomy of negotiation topics they prepared in the first step. Then, working together, they drag-and-drop every win condition into the taxonomic category to which it belongs. This typically takes two or three minutes. Next, the team opens each bucket to review its contents. They ask themselves these questions:

- Is there anything in this bucket that belongs in a different bucket? (If so, the team talks about it, then moves the win condition to a new bucket).
- Is there anything in here that belongs in a bucket we don't have yet? (If so, the team adds a new category to the taxonomy, then moves the win condition into that bucket).
- Are there any win conditions missing from this category (If so, the team captures the new win conditions in the WinWin tree, then cycles back for a new round of Issues, Options, and Agreements).

If one or more categories are insufficiently populated the team loops back into another iteration of the EasyWinWin process.

5. Conclusions

In the real world requirements emerge in a process of collaborative learning and negotiation. Each step in EasyWinWin challenges the stakeholders to use their tacit knowledge in a different way. The group dynamics created by the methodology helps the users to access tacit knowledge that might otherwise go untapped. This surpasses traditional ways of gathering requirements like interview-based techniques, which lack the ability to surface tacit knowledge.

Finally, notice that this methodology prevents people from arguing aloud about system requirements until the win conditions have been listed and explained, and assessed, until issues have been raised, and options have been proposed. When the team finally starts oral arguments, they tend to be very focused. The conversations do not ramble, digress, or go in circles. As the arguments unfold, their logic is captured in the

WinWin tree. Thus, when an agreement is reached, there is a record of how and why it was taken. As one stakeholder said, "EasyWinWin is more civilized way of negotiating requirements."

6. Acknowledgements

This work has been supported by the Austrian Science Fund (Erwin Schrödinger Grant 1999/J 1764 "Collaborative Requirements Negotiation Aids").

The authors would like to thank Barry Boehm (University of Southern California, Center for Software Engineering) for his great contributions and the collaboration during the EasyWinWin project.

7. References

[1] Boehm B., A spiral model of software development and enhancement, IEEE Computer, 21(5):61–72, 1988.

[2] Boehm B., Bose P., Horowitz E., Lee M.J., "Software Requirements as Negotiated Win Conditions," Proc. 1994 Intl. Conf. Rqts. Engineering, IEEE April 1994.

[3] Boehm B., In. H. Identifying Quality Requirement Conflicts. IEEE-SOFTWARE, 13(2):25ff, March 1996.

[4] Boehm B., A. Egyed, J. Kwan, D. Port, A. Shah, R. Madachy. Using the WinWin Spiral Model: A Case Study. IEEE Computer, 7:33–44, 1998.

[5] Boehm B., P. Grünbacher, "Supporting Collaborative Requirements Negotiation: The EasyWinWin Approach", Proc. International Conference on Virtual Worlds and Simulation VWSIM 2000, San Diego.

[6] Curtis B., H. Krasner, N. Iscoe. A Field Study of the Software Design Process for Large Systems. Communications of the ACM, 31(11):1268–1287, 1988.

[7] Denis A.R., A.R. Heminger, J.F. Nunamaker, D.R. Vogel, Bringing automated support to large groups: the Burr-Brown experience. Information & Management, 18, (1990), 111-121.

[8] Easterbrook S., Requirements Engineering: Social and Technical Issues, Resolving Requirements Conflicts with Computer-Supported Negotiation, S. 41–65. Academic Press Ltd, London, 1994.

[9] Fjermestad J., R. Hiltz, Case and Field Studies of Group Support Systems: An Empirical Assessment, Journal of Management Information Systems, in press.

[10] Gruenbacher P., Collaborative Requirements Negotiation with EasyWinWin, 2nd International Workshop on the Requirements Engineering Process, Greenwich, London, IEEE Computer Society, 2000.

[11] Gruenbacher P., EasyWinWin OnLine: Moderator's Guidebook, A Methodology for Negotiating Software Requirements. GroupSystems.com and USC-CSE 2000.

[12] Holtzblatt K., H. Beyer. Requirements Gathering: The Human Factor. Communications of the ACM, 38(5):31–32, 1995.

[13] Keil M., E. Carmel, Customer-Developer Links in Software Development, Communications of the ACM, 38(5):33–44, 1995.

[14] Macaulay L., Requirements Capture as a Cooperative Activity. In: Proceedings Of the First Intl. Symp. On Requirements Engineering, S. 174–181. IEEE Press, San Diego, CA, January 1993.

[15] McDermid J., Requirements Engineering: Social and Technical Issues, Requirements analysis: Orthodoxy, fundamentalism and heresy, S. 19–40. Academic Press, 1994.

[16] Muller P., J. van Engelen, C. Kappert, P. Terlouw. Facilitating Communication of New Product Development through Cooperative, Computer augmented Concept development. Computers in Engineering Symposium at ASME ETCE 96, I:30–40, 1996.

[17] Nissen H., M. Jeusfeld, M. Jarke, G. Zemanek, H. Huber. Managing Multiple Requirements Perspectives with Metamodels. IEEE-SOFTWARE, 13(2):37ff, March 1996.

[18] Nonaka I., A dynamic theory of organizational knowledge creation, Organization Science, vol. 5, 14-37, 1994.

[19] Nunamaker, J., R. Briggs, D. Mittleman, D. Vogel & P. Balthazard, Lessons from a Dozen Years of Group Support Systems Research: A Discussion of Lab and Field Findings, Journal of Management Information Systems, Winter 1996-97, 13(3), pp.163-207.

[20] Nuseibeh B., S. Easterbrook. Requirements Engineering: A Roadmap, In: The Future of Software Engineering, Special Issue 22nd International Conference on Software Engineering, ACM-IEEE, pp. 37—46, 2000.

[21] Nuseibeh B., Conflicting Requirements: When the customer is not always right. Requirements Engineering, 1/1:70–71, 1996.

[22] Post B.Q., A business case framework for group support technology, Journal of Management Information Systems, 9,3 (1993), 7-26.

[23] Standish Group CHAOS Report: Application Project and Failure, 1995.

[24] F. Varela, E. Thompson, E. Rosch. The Embodied Mind: Cognitive Science and Human Experience. MIT Press: Cambridge, 1992.

[25] L. Wittgenstein. "Schriften". Suhrkamp: Frankfurt, 1960.