EasyWinWin: Managing Complexity in Requirements Negotiation with GSS

Robert O. Briggs
GroupSystems.com – University of Arizona
1430 E. Fort Lowell Rd. #301, Tucson, AZ
bbriggs@groupsystems.com

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

Abstract

More than ¾ of large software projects suffer large cost and schedule overruns or fail outright. Deficits in project requirements cause more than half of these failures and overruns. This is in part because the establishing of software requirements is fraught with complexity. Finding ways to manage that complexity might be an important step in reducing the risk of software development. Group Support Systems (GSS) offer functionality that may reduce some aspects of complexity and reduce the cognitive load of addressing other aspects of complexity. In this paper we examine the sources of causes of complexity in software requirements in the context of EasyWinWin, a requirements negotiation methodology supported by GSS. Early field trials suggest that EasyWinWin is a significant step forward in managing the complexity of establishing requirements, and that further advantage could be gained by combining a GSS solution with other technologies like intelligent agents and requirements management systems.

1. Introduction

Large-scale software development projects are risky. More than a quarter of all software engineering projects fail outright. Of the rest, more than 70% suffer the doubling of budgets and schedules [15]. These failures waste hundreds of billions of dollars per year, so developers and consumers of software might derive significant value from any interventions that could reduce their risks.

More than half of these overruns and failures in software development projects can be directly attributed to flawed requirements [15, 16], so there is high potential for reducing the risk of software development by improving the processes by which requirements are established. One reason why software developers struggle to establish requirements is that the task is fraught with complexity. Finding ways to manage that complexity might be an important step in improving the quality of the requirements process.

In this paper, we draw on field experience gained during a two-year effort to develop and deploy new requirements negotiation methodology. We draw from and extend Wood's [17] model of task complexity to report some sources and causes of complexity that emerged during more than 50 requirements negotiation workshops we conducted during the project. We then argue that group support systems (GSS) can be useful to help manage the complexity inherent in requirements collection. We present the steps of EasyWinWin, a GSS-based requirements negotiation method that emerged from extensive user experiences, and discuss the manner in which each step addresses task complexity. We conclude with a discussion of future directions for using technology to address complexity in requirements tasks.

2. Complexity in Requirements Definition Tasks

Wood [17] identifies three sources and three causes of task complexity (Figure 1). All of these factors weigh heavily in software development projects. In Wood's model, the sources of complexity are 1) products (deliverables); 2) acts (behaviors required to create products); and 3) information cues (knowledge that permits actors to make judgments). The classes of complexity are the kinds of complexity that can manifest in any of these elements. They are: 1) Component complexity (number of and interdependency among acts and information cues needed to create products 2) Coordination complexity (the frequency, timing, intensity, and interdependencies of sequencing interactions required to produce products); and 3) Dynamic Complexity (the degree to which required products, acts, and information cues and the interdependencies among them change during the task).

Classes of Task Complexity

Sources of Task Complexity		Component	Coordination	Dynamic
	Products	Goals Use Cases Requirements	Derivative Dependencies	Change in vision
	Acts	Life Cycle Activities	Critical Path Dependencies	Changes in Process
	Information Cues	Identify and Resolve Conflicts Among Requirements	Model Clash	Changes in Meaning

Figure 1. Sources and classes of task complexity. Wood [17] argues that task complexity comes from three sources: Products, Acts, and Information Cues. *Products* are the deliverables that must be created during the task. *Acts* are the behaviors that give rise to the deliverables. *Information Cues* are the those things one must know to act in a manner that will produce the deliverables. Three different kinds of complexity can emerge from these sources: component complexity, coordination complexity, and dynamic complexity. *Component* complexity springs from the number of and interdependencies among the acts and information cues required to produce the products. *Coordination* complexity springs from the frequency, timing, and intensity of sequencing of interactions required to product the products. *Dynamic* complexity springs from the degree to which products, acts, and information cues change during a task. The cells in this table show examples of complexity in the software engineering arena.

There can be very high component complexity in the products for a requirements definition task. In a large-scale project like an Air Traffic Control system there are thousands of requirements, each of which must be tracked throughout the life of the project. Further, there can be a high degree of interdependency among requirements. That interdependency can take several forms (Figure 2). For example, the choice of software architecture might *depend on* the choice of network architecture. Choosing network architecture *constrains* ones choice of messaging protocols. A level-of-service requirement may *conflict with* a time-to-market requirement. A cross-platform requirement may *generalize* requirements that a new system operate on UNIX and Windows. Requirements for data communication through sockets, secure sockets, and HTTP *elaborate on* a requirement for multiple messaging channels.

Thus, the component complexity of the products of a requirements definition task can be quite high. This, in turn, causes high component and coordination complexity in the acts and information cues for the project because the project team must take action and marshal vast quantities of information cues to identify and address the interdependencies among project requirements.

Requirements definition tasks also tend to engender high dynamic complexity, as expressed by Boehm's Maxim:

You don't know the requirements until the project is done.

Boehm [2] argues that there is no complete, objective set of requirements out in the environment waiting to be discovered and written down. Rather, as a project proceeds, the project team learns more about what is desirable, what is possible, and what is acceptable, and the requirements evolve.

In requirements definition, a proliferation of semantic and consequential meanings increases the component and dynamic complexity of information cues. Different people use the same words to express very different concepts. Until those differences were made explicit, information cues will be overloaded, murky and confusing, which adds to the complexity of the task. Understandings of consequential meaning can likewise be diverse. To one stakeholder a decision to place a new system on the Internet might mean "universal access." To another it might mean, "security risk." Differences of consequential meaning may arise because of a lack of information, or because the issues in question are probabilistic rather than deterministic. They may spring from conflicting assumptions, or they may arise because different people have been attending to different cues in the environment.

Differences of consequential meaning during requirements definition may also spring from a fourth source of complexity that is not addressed by Wood's model: differences in vested interest. Different stakeholders want different things from a project. The various end-users may want different, even mutually exclusive features and functions. The customer may focus on low cost. Management might focus on delivering the project on time and under-budget. Even the same individual can have different, mutually exclusive requirements for a project. For example, a user might want both a web browser as client and off-line capability.

It falls to the software analysts and engineers to find a process whereby they can wade through all the sources and causes of complexity in requirements definition.

3. EasyWinWin – Managing Complexity with GSS

Group Support Systems (GSS) offer a partial solution to addressing the complexity inherent in requirements definition. A group support system is a collection of collaborative software tools that a team may use to focus and structure their mental effort as they work together toward a goal [13]. A team may use a GSS to create, sustain, and change patterns of group interaction in repeatable, predictable ways [13]. 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 [8]. 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% [7, 14]. (See [7] for an exhaustive compendium of GSS field research).

Using GSS tools, one can create a sequence of steps for a team to follow as they work on their task. During each step, the system displays one or more tools with which the team can generate, organize, and evaluate concepts and information. By using GSS, a team can significantly cut the cognitive load of communication and deliberation. Because GSS tools allow multiple people to work together to structure the products and information cues with which they wrestle, the cognitive load of addressing component complexity may be reduced. Because GSS tools allow for simultaneous contribution without turn-taking, the cognitive load associated with coordination complexity may be reduced. Because teams using GSS can read and respond to one another's contributions in real time, diversity of interests can be identified

Kinds of Interdependence	Degree	Definition
Depends On	n	A requirement *depends on* another when one requirement only makes sense if the other is in place.
Constrains	n^2	A requirement *constrains* another when choosing the first limits the options available for the second.
Conflicts with	n^2	A requirement *conflicts with* another when choosing the first precludes choosing the second.
Generalizes	n	A requirement *generalizes* other requirements when it expresses a high-level concept of which the others are instances.
Elaborates on	n	A requirement is an *elaboration of* another requirement when the first creates a more-specific instance of the second.

Figure 2. Interdependence Among Requirements. This table lists several ways in which system requirements may be related to or interdependent with one another. The volume and degree of interdependence determines the component complexity of the products for a requirements definition task. This, in turn, affects the component and coordination complexity of the acts and information cues for the task.

and accommodated early in the requirements process, which could reduce the need for changes later in the project. Because GSS allows many participants to review and annotate contributions, there may be increased opportunities to identify previously unnoticed interdependencies among requirements. Because GSS allows a team to focus and structure their interactions in predictable ways [6], GSS can become the foundation for developing and refining a repeatable, efficient requirements process.

We have developed a collaborative requirements method called EasyWinWin [11, 5] which is comprised of nine steps supported by GSS[1]. A key goal of the approach is to reduce the cognitive load associated with the sources and causes of complexity in the requirements definition task without losing or overlooking any of the richness of interrelationships among the many concepts incorporated in the requirements deliverables.

EasyWinWin combines the WinWin Spiral Model of Software Engineering [1, 2, 3] with collaborative knowledge techniques and automation of a Group Support System. In the WinWin negotiation model, the objectives of stakeholders are captured as *win condition*s. Conflicts among win conditions are recorded as *issue*s. *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.

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. 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 [12].

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.

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

The nominal purpose of the EasyWinWin methodology is to create an acceptable set of system requirements [4, 9, 10]. To fulfill their nominal purposes, each step is designed to manage one or more of the causes in one or more of the sources of task complexity.

The following sections summarize each step of the methodology and describe how task complexity is addressed during those steps.

Step 0. Engage the success-critical stakeholders. Boehm [5] argues that there is no objective, complete set of requirements "out there" waiting to be discovered and written down. Rather, he argues, requirements emerge from a process of learning and negotiation as people discover the financial and technical constraints under which they must work, and as they learn about one another's needs and interests. If that is the case, then one may reduce coordination complexity by involving only success-critical stakeholders in requirements negotiations. A success-critical stakeholder is any individual whose interests must be accommodated in order for the project to succeed. The success critical stakeholders are the people who can make agreements about the requirements, and make those agreements stick. If low-level representatives negotiate requirements, the success-critical stakeholder may subsequently disallow any agreements they reach. Such repudiation means more negotiations, which may again end with the repudiation of agreements by superiors. Having only success-critical stakeholders involved can short-circuit the negotiate-repudiate-renegotiate cycle, which should, in turn, reduce coordination complexity -- the frequency and intensity of interactions required to achieve the requirements. Therefore the first step of EasyWinWin is to identify and engage the participation of success-critical stakeholders.

Also, it is important to notice that success-critical stakeholders typically change throughout a project which increases dynamic complexity. For example, stakeholders negotiating a contract are different from stakeholders planning and performing the transition of a system to the target environment. The WinWin spiral model therefore demands the identification of success-critical stakeholders whenever a new cycle is entered.

Step 1. Refine and Expand Negotiation Topics. One difficulty created by the component complexity of software requirements is that most stakeholders are unaware of all the different aspects of a system for which requirements must be written. Therefore, in this step, the system presents the stakeholders with a shared outline. The outline contains a taxonomy of system requirements. The nodes of the outline are presented as categories for the many ways to win during a software development project. 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 with respect to task complexity. First, it addresses component complexity of requirements. Stakeholders tend to arrive with a narrow understanding of what they want and need from the proposed system. By reviewing and revising the taxonomy, they often come to understand that the project is much bigger than they had imagined, that the task will produce more products requiring more coordination than they had originally expected.

This step also reminds the participants to consider many concepts they might otherwise overlook. For example, a stakeholder from management who is primarily concerned about budgets and schedules might not think to state win conditions about interface response times.

Stakeholders prune some parts of the taxonomy and elaborate others. Any change that one stakeholder makes to the outline displays immediately on the screens of the other stakeholders interdependencies among elements of the project emerge and are recorded. In subsequent steps the team addresses the one by one. 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 step is also the first shot across the bow of dynamic complexity, because the better the stakeholders understand the scope of their task early in the project, the fewer the changes that may be required later in the project.

Step 2. Brainstorm Stakeholder Win Conditions. Stakeholders often arrive with at least a vague understanding of what they want from the system for themselves and their constituents. However, they often have little knowledge of what other stakeholders want from the system. Complexities of interest can only be addressed when stakeholders understand one another's interests. This step accomplishes three main purposes.

1. Stakeholders record first-draft statements of what they want from the proposed system
2. Stakeholders learn what others want from the system
3. Stakeholders expand and clarify what they want from the system by reading what others want

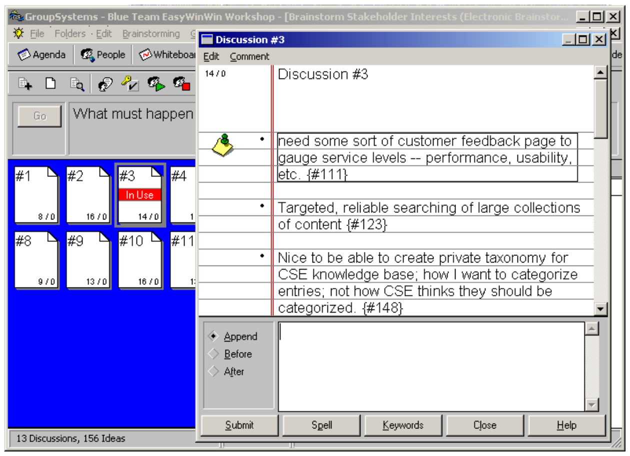
In this step, the stakeholders use an electronic brainstorming tool to surface as many different possible win conditions as they can in a short period of time (See Figure 3 for more detail about how this step is conducted). A team of 10 stakeholders typically contributes about 300 ideas in an hour-long brainstorm. Here are three examples of brainstorming comments submitted by stakeholders in a requirements negotiation about a knowledge management system:

Nice to be able to create private taxonomy for CSE knowledge base; how I want to categorize entries; not how CSE thinks they should be categorized.

No special knowledge is required for creation of cross-references.

Authorized users (without having to learn and use special applications) publish content in the intranet

This step can reduce coordination complexity for the software analyst or engineer charged with establishing requirements. Rather than interviewing stakeholders one-on-one or in small groups, many stakeholders can be brought together to contribute simultaneously, which reduces the frequency and intensity of interactions required. This step does not reduce component complexity, rather, it provides a means of managing it. Indeed, since this approach generates more requirements than other methods we have tried, it might appear to increase component complexity. However, experience suggests that the win conditions not revealed by standard methods still exist, and surface later in the project, which introduces the need for change. By surfacing requirements earlier in the project, there is the potential to reduce dynamic complexity.



Discussion #3

- need some sort of customer feedback page to gauge service levels -- performance, usability, etc. {#111}
- Targeted, reliable searching of large collections of content {#123}
- Nice to be able to create private taxonomy for CSE knowledge base; how I want to categorize entries; not how CSE thinks they should be categorized. {#148}

Figure 3. Brainstorm stakeholder interests. In this step, there is an electronic page for each stakeholder. Each time a stakeholder contributes a comment to a page the system takes that page away and randomly replaces it with a different page containing comments from other stakeholders. As the activity progresses, the pages swap among the participants, picking up a new comment at each stop. This process tends to broaden the scope of the discussion, resulting in breadth, rather than depth. It is a useful way to identify many concepts in a short amount of time.

Step 3. Converge on Win Conditions. To minimize component complexity, it is important to minimize the number of artifacts with which the team must deal. The contents of the brainstorming session in the previous step tend to be free ranging, wordy, full of redundancy and irrelevancy. Therefore, in this step, the team converges on a concisely worded, 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 component complexity for stakeholders by allowing them to work in parallel on smaller chunks of their data. 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 a win condition to be posted on the shared list. When each stakeholder has contributed one win condition to the shared list, the system reshuffles the raw brainstorming comments so each person now sees a different set. Stakeholders review that set to see if they can identify new win conditions. They continue to swap raw brainstorming comments and post new win conditions to the shared list until nobody can find anything new to add. The group discusses each win condition aloud to create a shared understanding of its semantic meaning. Participants may argue about the meaning of any win condition, but they may not object to or raise issues about any win condition at this time. This manages the component and coordination complexity surrounding the requirements by explicitly precluding discussions of consequential meaning and interdependency. 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 keyword 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, so component complexity has usually been reduced by this step.

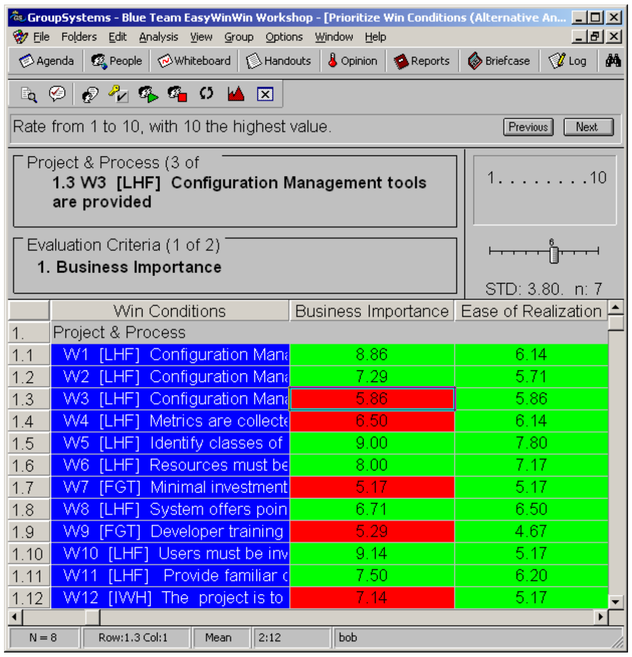
Step 4. Define a Glossary of Key Terms. In any system development project there are key terms that become insider jargon for project members. Insider jargon can simplify communication among those who know the jargon, but it can hinder communication with others who do not know the jargon. This step captures 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 additional 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. This can significantly raise the coordination complexity of information cues. In one recent case, the term, “affiliate” turned out to have five different meanings. When the team got to this step, each of those meanings was given a new, more specific term, and those five terms were added to the list. The group agreed not to use the term, “affiliate” for the rest of the project to minimize confusion.

Other times it turns out that several key terms have the same meaning. The team chooses one label for the concept and deletes all other redundant labels. This reduces the component and coordination complexity of information cues. 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. During brainstorming, convergence, and definitions of key terms, the stakeholders can post any win condition that comes to mind, regardless of its potential impact on other win conditions. Stakeholders learn about one another’s interests, but not necessarily about how important one win condition is compared to another, nor about what a given win condition might cost in time, effort, and aggravation. In this step, the participants rate each win condition along two criteria: (a) Business Importance - the degree to which the success of the project depends on this win condition being realized, and (b) Ease of Realization - the degree to which a win condition is technologically, socially, politically, and economically feasible (Figure 4).

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 merits of the win conditions. However, the results are not used to eliminate any win conditions. Rather they are used to provoke a well-structured, tightly 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.



Rate from 1 to 10, with 10 the highest value.

Project & Process (3 of
1.3 W3 [LHF] Configuration Management tools are provided

Evaluation Criteria (1 of 2)
1. Business Importance

STD: 3.80. n: 7

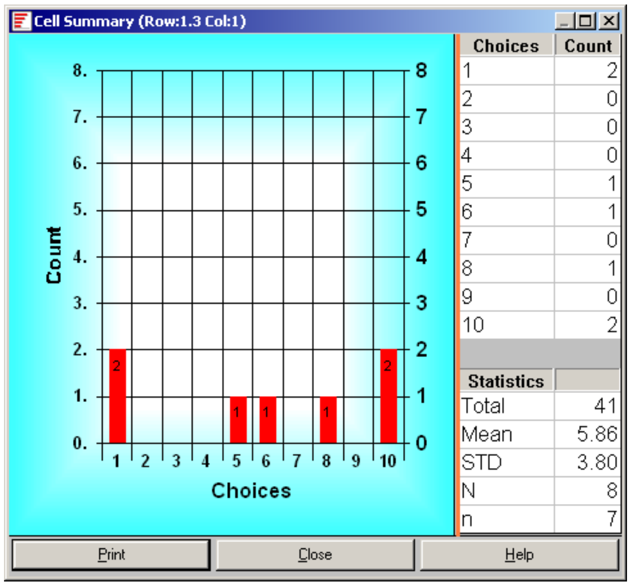
	Win Conditions	Business Importance	Ease of Realization
1.	Project & Process		
1.1	W1 [LHF] Configuration Mana	8.86	6.14
1.2	W2 [LHF] Configuration Mana	7.29	5.71
1.3	W3 [LHF] Configuration Mana	5.86	5.86
1.4	W4 [LHF] Metrics are collecte	6.50	6.14
1.5	W5 [LHF] Identify classes of	9.00	7.80
1.6	W6 [LHF] Resources must be	8.00	7.17
1.7	W7 [FGT] Minimal investment	5.17	5.17
1.8	W8 [LHF] System offers poin	6.71	6.50
1.9	W9 [FGT] Developer training	5.29	4.67
1.10	W10 [LHF] Users must be inv	9.14	5.17
1.11	W11 [LHF] Provide familiar c	7.50	6.20
1.12	W12 [IWH] The project is to	7.14	5.17

N = 8 | Row:1.3 Col:1 | Mean | 2:12 | bob

Figure 4. Prioritizing Win Conditions. Participants use a multi-criteria polling tool to assess the business importance and ease of implementation for each win condition. The items where the group has high consensus display in green. The items where consensus is low display in red.

This step helps manage component and coordination complexity of products and information cues by explicitly excluding consideration of interdependencies while considering the merits of each condition as a stand-alone product. The team explores interdependencies in a separate step later in the process. It also helps manage component and coordination complexity for stakeholders by asking them to respond only to those items about which they have knowledge and interest. They need not consider the other items.

Step 6. Surface Issues and Constraints. Any given win condition may, on its own, raise issues for any given stakeholder. The purpose of the previous step was not to eliminate low-rated win conditions, but rather to surface differences of opinion about individual win conditions. Different stakeholders often have different reasons for the opinions they register, and those reasons spring from their differences of experience, interest, and purpose. Those differences often relate to unarticulated and unexamined project constraints. This step focuses exclusively on the areas of highest disagreement among the ballots cast in the previous step. When the results are displayed, items with high consensus display with a green background, while items with low consensus display with a red background (Figure 4). This step focuses on the red cells, where consensus is low. A click on a red item displays a graph and a table that reveals the pattern of votes underlying that cell (Figure 5).



Choices	Count
1	2
2	0
3	0
4	0
5	1
6	1
7	0
8	1
9	0
10	2

Statistics	
Total	41
Mean	5.86
STD	3.80
N	8
n	7

Figure 5. Polling Patterns Under a Red Cell. Stakeholders use this graph as a stimulus to explore the reasons behind their differences of opinion about a win condition.

The group holds a structured oral 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. Key information cues about the project emerge from these discussions

– 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 bimodal split in the group. It turned out that some stakeholders assumed an external group that forbade the use of CGI-scripts would manage the system. 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.

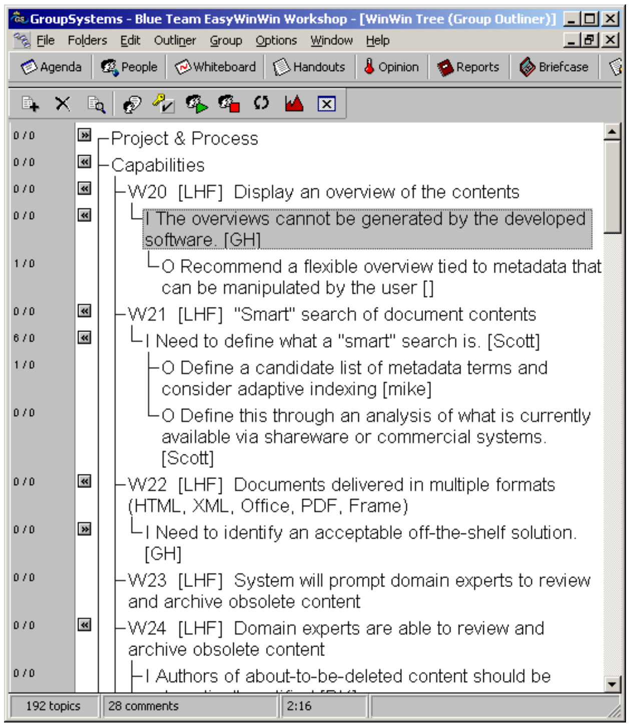
– Hidden agendas – captured as new win conditions.

The most important purpose of this step is to identify and address interest complexity. Sometimes differences of opinion emerge based on orthogonal interests – "It's important to you, but I simply don't care, so I said it wasn't important." Other times the differences are based on mutually exclusive interests – "It's important to you to have it, but it's important to me to NOT have it". Such conflicts are identified and flagged for later negotiation, but to minimize coordination complexity of acts and information cues, no negotiation is allowed during this step.

Step 7. The WinWin Tree: Win Conditions, Issues, Options, Agreements. Any win condition may have interdependencies with other win conditions. One key purpose of this step is to identify and deal with those issues. This step also allows stakeholders to argue their case against any given win condition, should they have an issue with something proposed by someone else. In this step, the team posts a shared outline with all the win conditions as main headings (Figure 6). 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.

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. 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, the group usually declares these items to be 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, more assumptions and constraints, more key terms, more options, more issues, and more win conditions emerge. 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 the agreement on the WinWin Tree. When every win condition and every option has an agreement, the state of WinWin Equilibrium has been achieved.


GroupSystems - Blue Team EasyWinWin Workshop - [WinWin Tree (Group Outliner)]
File Folders Edit Outliner Group Options Window Help
Agenda People Whiteboard Handouts Opinion Reports Briefcase

```
0/0   -Project & Process
0/0   -Capabilities
0/0     -W20  [LHF]  Display an overview of the contents
0/0       └I The overviews cannot be generated by the developed
           software. [GH]
1/0         └O Recommend a flexible overview tied to metadata that
             can be manipulated by the user []
0/0     -W21  [LHF]  "Smart" search of document contents
6/0       └I Need to define what a "smart" search is. [Scott]
1/0         -O Define a candidate list of metadata terms and
             consider adaptive indexing [mike]
0/0         └O Define this through an analysis of what is currently
             available via shareware or commercial systems.
             [Scott]
0/0     -W22  [LHF]  Documents delivered in multiple formats
         (HTML, XML, Office, PDF, Frame)
0/0       └I Need to identify an acceptable off-the-shelf solution.
           [GH]
0/0     -W23  [LHF]  System will prompt domain experts to review
         and archive obsolete content
0/0     -W24  [LHF]  Domain experts are able to review and
         archive obsolete content
0/0       -I Authors of about-to-be-deleted content should be

192 topics | 28 comments | 2:16
```

Figure 6. The WinWin Tree. Participants begin with an outline of their win conditions. They make three passes through the outline. On the first pass they read each win condition, and if they have an issue with any win condition, they record it as a subheading to the win condition. On the next pass, they read every issue, and if they can think of an option for resolving the issue, they record it as a subheading to the issue. On the next pass they negotiate agreements for every issue and every win condition.

The most important reason for this step is to manage the component and coordination complexity associated with the interests and purposes of the stakeholders. Without any objection or impediment from others, every stakeholder may raise and explain any issues with any win condition posted by other stakeholders. Because all are contributing simultaneously, rather than in an oral discussion, their issues are captured quickly, with digression into interpersonal conflict. Before they are allowed to negotiate any issues, however, every team member has the opportunity to propose solutions for the issues. So when the oral negotiations begin, there are usually useful ideas already proposed. The discussion focuses on a single issue at a time, which keeps the negotiations manage-able. If someone raises a new issue during the discussion, that issue is recorded on the WinWinTree for later negotiation, and discussion returns to the issue at hand.

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.

The primary purpose of this step is to manage component complexity. The requirements for a large system are numerous. This step gives the team the opportunity to check whether they have addressed all important topics in their negotiations.

4. User Experiences

EasyWinWin has been used in about 50 projects so far. The approach has been validated in typical software development projects (e.g., digital library projects, extension of COTS products, web portals for e-marketplaces) and has been applied in different stages of the life-cycle (e.g., to build a shared vision among the stakeholders, to develop high-level requirements for a project, to learn more about the requirements for transitioning a system to the target environment).

Since effective negotiation turns out to be an important success-factor not only in software engineering activities our approach has been adopted in other areas as well: One organisation used EasyWinWin to support action planning during a process improvement initiative, another organization used the methodology to jointly develop and negotiate the requirements for their marketing processes. We have found that the EasyWinWin process is extremely helpful to structure the negotiation activities, and at the same time allows a team to handle a much higher volume of information than a traditional paper- or blackboard-based negotiation process would. Typical negotiations we have carried out involved 10+ participating stakeholders resulting in 300+ brainstorming ideas, 100+ win conditions, 50+ issues, 50+ issues, and 100+ agreements.

5.Future Directions

Early field results suggest that EasyWinWin does, in fact help teams manage the complexity of requirements negotiations for large software development projects. Requirements projects using EasyWinWin typically require weeks rather than months to complete, and the resulting requirements have an order of magnitude increase in detail. However, *experience in the field suggests that* further advantage could be gained by combining *the* GSS *approach* with other technologies.

Perhaps the most difficult challenge still facing stakeholders is the identification of conflicts among win conditions. Any win condition dealing with time-to-market, for example, may conflict with other win conditions dealing with quality. Any win condition dealing with features and functions may conflict with win conditions for cost containment. The interdependencies among win conditions can be many, subtle, and varied. It might be possible that intelligent agents could be used to identify and flag potentially conflicting win conditions, and to bring them to the attention of stakeholders.

It would also be useful if analysts could move the win conditions seamlessly from the GSS in which they were developed into a requirements management system like Rational's RequisitePro or Telelogic's DOORS. That way, as technical requirements were written, they could be traced back to the win conditions they were meant to address. That way, as the project progresses, when it becomes necessary to change technical requirements, analysts and engineers could know why a requirement emerged in the first place, and whose interests it serves. They may be better positioned to develop solutions with which the stakeholders can live, and they would know better which stakeholders should be involved in negotiating the proposed changes.

The component complexity of win conditions still posses a significant challenge to EasyWinWin workshop participants. Research now underway by Hoh In and others seeks to assist with that challenge by using intelligent agents to identify and flag potential conflicts among win conditions. These researchers seek to embed heuristics into an ontology that the agent can use to find conflicts that humans might overlook. For example, a win condition that deals with quality might conflict with other win conditions that deal with time-to-market. Intelligent agents may be able to conduct exhaustive pair-wise comparisons among thousands of win conditions, a task that would be impossible for individual humans.

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

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

[5] Boehm, B; Gruenbacher, P. , & Briggs, R. O. Developing Groupware for Requirements Negotiation: Lessons Learned. IEEE Software, Vol. 18, No. 3, May/June 2001

[6] Briggs, R.O.; de Vreede, Gert-Jan; Nunamaker, J.F., and Tobey, David H. ThinkLets: Achieving Predictable, Repeatable Patterns of Group Interaction with Group Support Systems (GSS) Proceedings of the 34th Hawaii International Conference on System Sciences, 2001.

[7] Dennis 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] Fjermestad J., R. Hiltz, Case and Field Studies of Group Support Systems: An Empirical Assessment, Journal of Management Information Systems, 2000-01.

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

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

[11] Gruenbacher, Paul and Briggs, R.O. Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin Proceedings of the 34th Hawaii International Conference on System Sciences, 2001.

[12] 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,

[13] 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,

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

[15] Standish Group CHAOS Report: Application Projects and Failures, 1995.

[16] Standish Group CHAOS Chronicles II. 2001.

[17] Wood, R.E. Task complexity: definition of the construct. *Organizational Behavior and Human Decision Processes,* 37(1), 1986, 60-82.