Developing Groupware for Requirements Negotiation: Lessons Learned

Barry Boehm, *University of Southern California*
Paul Grünbacher, *Johannes Kepler University Linz*
Robert O. Briggs, *GroupSystems.com*

The authors discuss the lessons they learned while developing four generations of a distributed groupware system called WinWin.

Defining requirements is a complex and difficult process, and defects in the process often lead to costly project failures.[1] There is no complete and well-defined set of requirements waiting to be discovered in system development. Different stakeholders—users, customers, managers, domain experts, and developers—come to the project with diverse expectations and interests. Requirements emerge in a highly collaborative, interactive, and interdisciplinary negotiation process that involves heterogeneous stakeholders.

At the University of Southern California's Center for Software Engineering, we have developed a series of groupware implementations for the WinWin requirements negotiation approach (see the Acknowledgments at the end of the article for a list of organizations that helped sponsor this research). The WinWin approach involves having a system's success-critical stakeholders participate in a negotiation process so they can converge on a mutually satisfactory or win–win set of requirements. Groupware-supported methodologies are among the hardest to get right, and the rapidly moving technology of distributed interactive systems is a major challenge. This is due largely to the relative newness of the area and to the unfamiliarity of most software developers with the phenomena of group dynamics. However, an even bigger challenge is creating a system that works well with people of different backgrounds, in different places, and often at different times. In particular, collaborative technology that supports requirements negotiation must address stakeholder heterogeneity.

Our WinWin groupware system—which has evolved over four generations—enables and facilitates heterogeneous stakeholder participation and collaboration. Each generation reflects an increase in our understanding of what is needed for successful WinWin groupware operations and technology support. Here, we present the major lessons we learned during WinWin's development.

0740-7459/01/$10.00 © 2001 IEEE

The WinWin approach

The original motivation for a WinWin groupware system was Barry Boehm's frustration in using a manual win–win approach to manage large projects at DARPA. For example, win–win management of the $100-million DARPA STARS program was done primarily through monthly meetings of many STARS stakeholders: three prime contractors and their three commercial counterparts; three user representatives from the Army, Navy, and Air Force; DARPA customers and contract managers; and several research and support contractors. Each meeting concluded with a win–win agreement, so after a meeting, participants felt they had taken three steps forward. However, by the next meeting, the distributed stakeholders had independently "reinterpreted" the agreements, causing the process to move two steps back. As a result, it took six months to achieve a shared vision that the prime contractors' success plans documented. Our analysis at the time indicated that a WinWin groupware support system could reduce this process to one or two months.

The general win–win approach evolved more or less independently as an interpersonal-relations,[2] success-management,[3] and project-management[4] approach. We usually define it as "a set of principles, practices, and tools, which enable a set of interdependent stakeholders to work out a mutually satisfactory (win–win) set of shared commitments."

Interdependent stakeholders can be people or organizations. Their shared commitments can relate to information system requirements in particular (the WinWin groupware system's primary focus) or can cover most continuing relationships in work and life (for example, international diplomacy). *Mutually satisfactory* generally means that people do not get everything they want but can be reasonably assured of getting whatever it was to which they agreed. *Shared commitments* are not just good intentions but carefully defined conditions. If someone has a conditional commitment, he or she must make it explicit to ensure all stakeholders understand the condition as part of the agreement.

Why does WinWin work?

WinWin works because people and groups have different preference patterns. A classic example was the 1978 Egyptian-Israeli peace treaty's negotiation of the Sinai Peninsula borderline. It was at an impasse until it was clarified that Egypt preferred territory and Israel preferred getting a demilitarized zone. We elaborate on other reasons why WinWin works in the following.

Win–lose doesn't work

In requirements negotiation, nobody wants a lose–lose outcome. Win–lose might sound attractive to the party most likely to win, but it usually turns into a lose–lose situation. Table 1 shows three classic win–lose patterns among the three primary system stakeholders—developers, customers, and users—in which the loser's outcome usually turns the two "winners" into losers.[5]

As the table shows, building a quick and sloppy product might be a low-cost, near-term win for the software developer and customer, but the user (and maintainer) will lose in the long run. In addition, adding lots of marginally useful bells and whistles to a software product on a cost-plus contract might be a win for the developer and users, but it is a loss for the customer. Finally, "best and final offer" bidding wars that customers and users impose on competing developers generally lead to lowball winning bids, which place the selected developer in a losing position.

However, nobody really wins in these situations. Quick and sloppy products destroy a developer's reputation and have to be redone—inevitably at a higher cost to the customer. The bells and whistles either disappear

In particular, collaborative technology that supports requirements negotiation must address stakeholder heterogeneity.

Table 1

Frequent Software Development Win–Lose Patterns (That Usually Turn into Lose–Lose Situations)

Proposed solution	"Winner"	Loser
Quickly build a cheap, sloppy product	Developer and customer	User
Add lots of "bells and whistles"	Developer and user	Customer
Drive too hard a bargain	Customer and user	Developer

A WinWin approach builds a shared vision among stakeholders and provides the flexibility to adapt to change.

or (worse) crowd out more essential product capabilities as the customer's budgets are exhausted. Inadequate lowball bids translate into inadequate products, which again incur increased customer costs and user delivery delays to reach adequacy.

WinWin builds trust and manages expectations. If you consistently find other stakeholders asking about your needs and acting to understand and support them, you will end up trusting them more. In addition, if you consistently find them balancing your needs with other stakeholders' needs, you will have more realistic expectations about getting everything you want.

WinWin helps stakeholders adapt to changes. Our traditional, adversarial, lawyer-oriented contracting mechanisms are no match for our current world of increasingly rapid change in technology, mergers, reorganizations, and personnel turnover. Instead of rigorous requirements in ironbound contracts, doing business in Internet time requires stakeholders with a shared vision and the flexibility to quickly renegotiate a new solution once unforeseen problems or opportunities arise.[6–7] A WinWin approach builds a shared vision among stakeholders and provides the flexibility to adapt to change.

WinWin helps build institutional memory. The why behind the what—that is, the decisions that lead to a work result—often vanish. By capturing and preserving stakeholder negotiations, WinWin supports long-term availability of the decision rationale and thus helps build institutional memory. Having more auditable decisions creates more detailed, accurate, and complete deliverables.

How does the WinWin negotiation model work?

The particular approach we have evolved includes a WinWin negotiation model for converging to a win–win agreement and a WinWin equilibrium condition to test whether the negotiation process has converged.

The negotiation model guides success-critical stakeholders in elaborating mutually satisfactory agreements. Stakeholders express their goals as win conditions. If everyone concurs, the win conditions become agreements. When stakeholders do not concur, they identify their conflicted win conditions and register their conflicts as issues. In this case, stakeholders invent options for mutual gain and explore the option trade-offs. Options are iterated and turned into agreements when all stakeholders concur. Additionally, we use a domain taxonomy to organize WinWin artifacts, and a glossary captures the domain's important terms. The stakeholders are in a WinWin equilibrium condition when the agreements cover all of the win conditions and there are no outstanding issues.

Four generations of tool support

The WinWin negotiation model provided the basis of all four generations of WinWin groupware systems.

G1: Initial prototype

The first-generation WinWin groupware implementation was a prototype developed in concert with Perceptronics' CACE-PM support system for concurrent engineering of multichip modules. CASE-PM let us develop a useful prototype in several weeks, which was sufficient for demonstrations and an initial experiment. This involved having the G1 WinWin system developers perform role-playing as future system developers, customers, and users negotiating the requirements for a more robust version of WinWin. Performing the WinWin negotiation with G1 WinWin gave us a strong, shared vision for the system's next version, validating its utility as a groupware capability.

G2: Strong vision, not-so-strong architecture

The second-generation WinWin system used a Sun-Unix client-server architecture, X/Motif GUI support, and its own database server. Some friendly industry users tried it experimentally. G2 WinWin's main value was identifying inconsistencies between the negotiation model and the artifacts, among the artifacts, and between the GUI and the database server. However, we underestimated how much detailed software engineering it would need to get from a shared groupware vision to a groupware support system.

G3: Muscle-bound architecture

The third-generation WinWin system had a formally analyzed negotiation model, a uniform artifact look and feel, carefully defined GUI-database interfaces, and rigorous enforcement of the negotiation model.

What Is a Group Support System?

Some group tasks can be most effectively accomplished by carefully coordinated individual efforts. Technologies that support this kind of teamwork abound—email, team calendaring, shared document repositories, and so on. Other tasks, such as requirements negotiation, require concerted reasoning by many minds. For such tasks there are Group Support Systems.

On the surface, a GSS might seem like a collection of glorified chat tools with some voting tools thrown in for good measure. For example, most GSS suites include shared list tools. Any user can make a contribution to a shared list at any time, and any contribution a person makes appears instantly on all the other users' screens. Various GSS suites include shared outlines, shared comment windows, shared drawing tools, and so on. In each tool, all the users can talk at once, contributing to the discussion as inspiration strikes rather than waiting for the floor. GSS suites usually include a variety of useful voting tools, including Likert scales, semantic anchors, allocation votes, and multicriteria votes. The users can move their contributions into a vote tool, evaluate them, and then instantly review their results online.

The real magic of a GSS is not what you can make happen on the screen but what you can make happen in the group. Using a GSS, you can create predictable, repeatable patterns of human interaction and reasoning among people working toward a common goal. For example, most GSSs include brainstorming tools that can help a group diverge from customary patterns of thinking. Some GSSs also have idea-organizing tools that let a group structure disorganized ideas. Other GSS tools can help a group converge quickly from its many brainstorming ideas down to a clear focus of just the ideas that merit further attention.

Using a GSS, you can arrange a sequence of steps that a team can follow as they reason together to accomplish their task. In each step, the GSS tools are configured so that as participants contribute to the system, a useful pattern of thinking emerges.

In all, there are seven basic patterns of thinking a GSS can create in a group:[1]

- Diverge: Move from having fewer ideas to more ideas,
- Converge: Move from having many ideas to focusing on just the few that are worthy of further attention,
- Organize: Start to gain a better understanding of the relationships among ideas,
- Elaborate: Start expressing ideas in more detail,
- Abstract: Move from expressing ideas in detail to expressing them as fewer, more general concepts,
- Evaluate: Start better understanding the value of concepts for accomplishing the task at hand, and
- Build consensus: Start better understanding the diverse interests of the group members and begin to agree on possible courses of action.

Besides requirements negotiation, GSS processes have been implemented for a variety of organizational tasks that require many people to think together. Here are a few examples:

- strategic planning,
- new product development,
- marketing focus groups,
- total quality management,
- military intelligence analysis,
- organizational change management,
- data modeling,
- group therapy,
- factory floor design, and
- software inspections.[2]

Because a GSS operates over a computer network, team members can often interact, even when oceans and continents separate them. However, just as a screwdriver is not very useful for pounding nails, a GSS is not right for every group interaction. Sometimes a team still needs to get together the old-fashioned way, eye-to-eye, to see who sweats and who blinks first. Nonetheless, extensive research shows that under the right circumstances, teams using a GSS can reduce their labor hours by 50 percent or more and cut their project cycles by 60 to 90 percent. Such teams usually also report a higher-quality result than they were able to obtain using more conventional means.[3]

References

1. R.O. Briggs, G.-J. de Vreede, and J.F. Nunamaker, Jr., "Thinklets: Achieving Predictable Repeatable Patterns of Group Interaction with Group Support Systems (GSS)," *Proc. HICSS 2001 (Hawaii Int'l Conf. System Sciences)*, IEEE CS Press, Los Alamitos, Calif., 2001.
2. M. van Genuchten et al., "Industrial Experence in Using Group Support Systems for Software Inspections," *IEEE Software*, vol. 18, no. 3, May/June 2001, pp. 60–65.
3. J. Fjermestad and R. Hiltz, "Case and Field Studies of Group Support Systems: An Empirical Assessment," *J. Management Information Systems*, to be published, 2001.

It also had a number of amenities for voting, for attaching associated documents or analysis-tool runs, and for big-picture negotiation visualization and navigation. Its major problems were its insufficient robustness and the overly strict enforcement of the negotiation approach that kept it from adapting to different negotiation situations.

G4: Group support system infrastructure

Our experiences with the first three generations of WinWin encouraged USC to develop a version of WinWin based on the commercial groupware infrastructure GroupSystems.com developed in cooperation with the University of Arizona.[8] Our current collaboration between USC and GroupSystems.com has led to a fourth-generation system, called EasyWinWin.[9]

EasyWinWin

EasyWinWin is a requirements definition approach based on a Group Support System. A GSS is a suite of software tools that can create, sustain, and change patterns of group interaction in repeatable, predictable ways

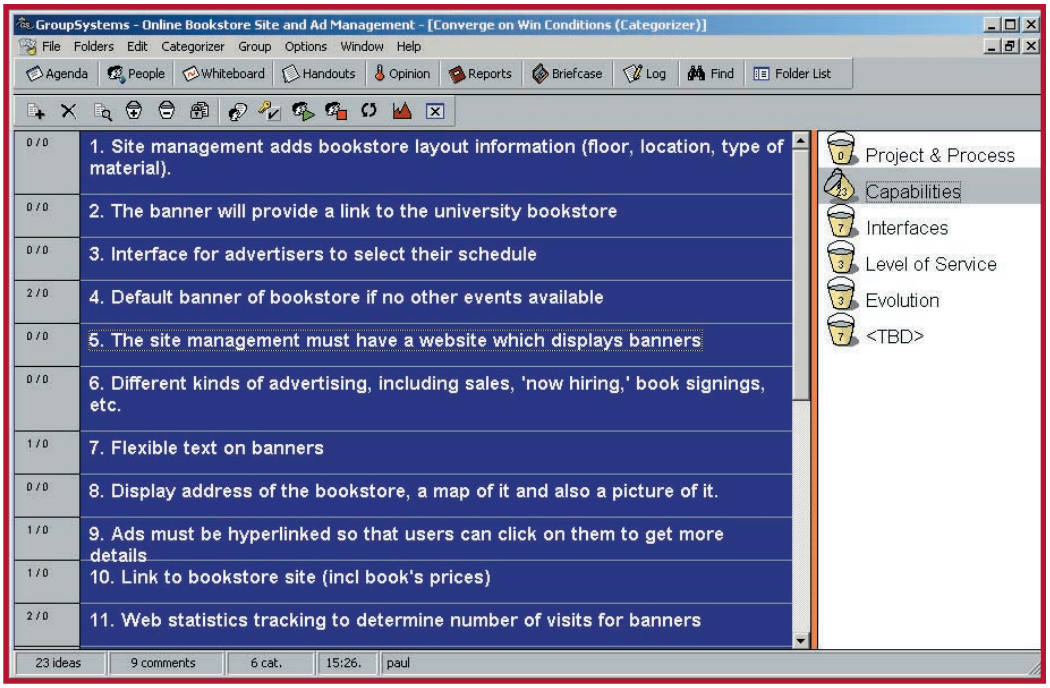

Figure 1. An example of how a team builds a clean list of win conditions and organizes them into predefined buckets.

(see the "What is a Group Support System?" sidebar for details). The tools help increase stakeholder involvement and interaction.

EasyWinWin defines a set of activities guiding stakeholders through a process of gathering, elaborating, prioritizing, and negotiating requirements. In addition, it uses group facilitation techniques that collaborative tools support.

Table 2 summarizes the main stakeholder negotiation activities involved in using EasyWinWin, and how they are implemented through group techniques and thinking patterns that the GroupSystems.com GSS supports. Figure 1 shows an EasyWinWin example of candidate win conditions for a USC bookstore Web portal requirements negotiation and how stakeholders can categorize them by dragging them into buckets. In contrast, 3G WinWin had a much more formal way of defining win conditions, in which categorization involved scrolling through a category list in another window and typing in the category name or number. With 3G WinWin, it would usually take about a day to enter and categorize about 15 to 25 win conditions. With EasyWinWin, it would take about two hours to enter and categorize

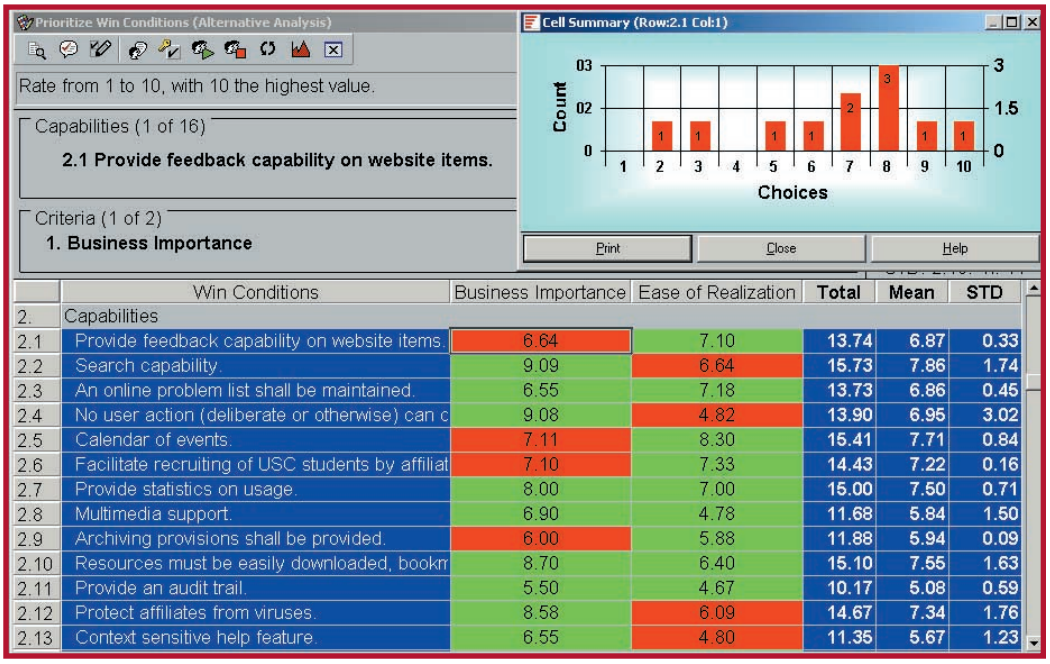

Figure 2. In EasyWinWin, red cells indicate a lack of consensus. The cell graph shows a voting pattern used to trigger an oral discussion revealing unshared information, unnoticed assumptions, hidden issues, constraints, and so forth. This particular graph shows the voting pattern for win condition 2.1, with criterion Business Importance.

Table 2

EasyWinWin Activities, Group Techniques, and Patterns of Thinking

(see the sidebar for information on patterns of thinking)

Activity	Purpose	Group technique	Thinking pattern
Review and expand negotiation topics	Stakeholders jointly refine and customize an outline of negotiation topics based on a taxonomy of software requirements. The shared outline helps stimulate thinking, organize win conditions, and check negotiations.	Stakeholders add comments and recommend changes to the outline. A moderator reviews these comments together with the group and modifies the outline	Diverge Converge
Brainstorm stakeholder interests	Stakeholders share their goals, perspectives, views, background, and expectations by gathering statements about their win conditions.	Free-format brainstorming: Anonymous, rapid brainstorming on electronic discussion sheets	Diverge
Converge on win conditions	Stakeholders jointly craft a nonredundant list of clearly stated, unambiguous win conditions by considering all ideas contributed in the brainstorming session.	Fast focus: A structured discussion to converge on key win conditions Categorize win conditions into negotiation topics.	Converge Organize
Capture a glossary of terms	Stakeholders define and share the meaning of important terms of the project/domain in a glossary of terms.	Stakeholders propose initial definition of terms based on stakeholder statements. The team then jointly reviews and agrees on the terms.	Elaborate
Prioritize win conditions	The team prioritizes the win conditions to define the scope of work and to gain focus.	Stakeholders rate win conditions for each of two criteria: business importance (relevance of a win condition to project/company success) and ease of realization (perceived technical or economic constraints of implementing a win condition).	Evaluate
Reveal issues and constraints	Stakeholders surface and understand issues.	Crowbar: Analyze prioritization poll to reveal conflicts, constraints, different perceptions, and so forth.	Build consensus
Identify issues, options, agreements	Identify the issues that arise owing to constraints and conflicting win conditions. Propose options to resolve these issues. Negotiate agreements.	WinWinTree: Review win conditions, identify issues, and porpose options Negotiation of agreements	Elaborate Build consensus

about 50 to 120 win conditions.

Figure 2 shows EasyWinWin's capability for group prioritization of win conditions. Stakeholders can quickly see which win conditions are more and less important and easy to implement, plus where their degree of consensus is stronger (the green and red cells). With 3G WinWin, we had an awkward interface to a separate and partially implemented prioritization tool that was rarely used.

EasyWinWin has been used in about 30 real-world projects. We applied the approach in various domains (for example, in digital libraries, an e-marketplace, and collaboration technology) and thoroughly explored and refined the various collaborative techniques with the goal of streamlining the negotiation protocols and the overall order and design of process steps. We captured our experiences in a detailed process guidebook that explains our approach to project managers or facilitators.[10]

Lessons learned

We learned three major types of lessons while developing the four generations of WinWin systems: methodology, groupware, and project.

Collaborative technology for requirements engineering must be based on collaboration and facilitation techniques that emphasize group dynamics.

Methodology lessons

Groupware tools for information technology requirements negotiation need a methodology that reflects both the evolving role of requirements in the IT life cycle process and the sensitivity involved in successful group dynamics. Here are some examples of methodology lessons we learned.

Define a repeatable requirements negotiation process. The first three generations provided only a top-level strategy on how to carry out a concrete WinWin negotiation. In the EasyWinWin project, we focused on moving people through a process that builds mutual understanding and a shared vision in progressive steps. Each step involves the stakeholders in assimilating each others' views and in building consensus on a mutually satisfactory shared vision and set of system requirements. A process guide explains the use of group techniques and collaborative tools in a requirements negotiation.[10] We found that a detailed process guide reduces variance in the quality of deliverables and helps lower-skilled or less experienced practitioners accomplish more than would be possible with straight stand-up facilitation.

Incorporate facilitation and collaboration techniques. Collaborative technology for requirements engineering must be based on collaboration and facilitation techniques that emphasize group dynamics. The first three generations of WinWin environments emphasized modeling constraints over group dynamics and collaboration support. Groupware-supported collaboration techniques adopted in EasyWinWin help create desired patterns of group interaction (see the sidebar).

An example is the anonymous submission of stakeholder contributions, such as win conditions or voting ballots used to foster candor. This way, people with power differentials can make proposals without feeling a threat to their job, status, relationships, or political position. Increased openness also helps stakeholders quickly get to the root issues. People up and down the hierarchy are better informed and thus can avoid the Abilene Paradox (in which people agree to an unattractive option because they erroneously believe it will make the option-proposer happy).[11]

Recognize the role of negotiations in the life cycle. A major lesson learned from the experiment with the 1G WinWin system was that the WinWin approach helps bridge a previous gap[5] in using the spiral process model: determining the next round of objectives, alternatives, and constraints. This led to the WinWin spiral model extensions that several organizations now use.[12]

Experiments also showed that we should perform prototyping ahead of and during requirements negotiations: The 3G WinWin was sufficiently robust to support four years' worth of projects—with 15 to 20 project negotiations per year.[12,13] These project negotiations involved USC librarians and student teams negotiating the requirements for operational USC digital library systems, which the student teams then built and transitioned to library use. In the first year, we learned not to do the WinWin negotiations ahead of the prototype, as we rediscovered the IKIWISI (I'll know it when I see it) syndrome. Once the librarians saw the prototype, they wanted to redo all the negotiations. In the following years, we verified across over 100 requirements negotiations that 3G WinWin could support rapid definition and development of unprecedented applications.

WinWin has been successfully used in various contexts of a requirements definition. This includes the development of a shared vision among stakeholders, requirements definition for custom development projects, COTS acquisition and integration, transition planning, and COTS product enhancement and release planning.

Make sure your stakeholder negotiators have the essential characteristics. Your stakeholder negotiators should be representative, empowered, knowledgeable, collaborative, and committed. We identified these characteristics after analyzing the critical success factors for transition into digital library operational use. Successful win–win negotiations often involve prescreening stakeholder negotiators and performing shared-knowledge-building activities such as preliminary team-building sessions and concurrent prototyping.

Refine agreements into more measurable requirements. The result of a WinWin negotiation is typically not a complete, consis-

tent, traceable, and testable requirements specification. For example, stakeholders might become enthusiastic about proposed capabilities and ratify idealistic agreements such as "anytime, anywhere" service. Rather than dampen their enthusiasm with a precise wordsmithing exercise, it is better to have a facilitator postprocess such agreements into more precise and realistic requirements for the stakeholders to review and iterate.

Groupware lessons

Some of the strongest agreements among negotiators in our early WinWin systems were about deficiencies in the groupware. This is not where you want your negotiators focusing their attention. Later versions of WinWin became more effective in keeping stakeholders focused on their negotiations.

Make use of unobtrusive and flexible tools. Each of our first three generations of WinWin groupware was increasingly strict about enforcing modeling conventions at the expense of group dynamics. Several industry and government organizations also used 3G WinWin experimentally. However, this use did not lead to the system's crossing the chasm into mainstream use. The main reason the users cited was that 3G WinWin's integrity rules were too rigorous. For example, when an agreement was put to a vote, all of its associated win conditions, issues, and options were locked to preserve the voting process's integrity. Then, to fix a typo in an artifact that was locked for voting, users had to make all the locked artifacts inactive and copy their contents into a new set of artifacts. Furthermore, they could not define issues without win conditions as referents or define options without issues as referents. Thus, the software got in the way of the human interactions, which were a critical part of negotiation. In EasyWinWin, we decided to relax such constraints, with the result that industry and government users have been enthusiastic about their negotiation experiences.

Define the negotiation model. Experiments with the 1G WinWin system showed that software requirements negotiation required considerably more database and relationship management than was needed for multichip modules. This led to a much more thorough definition of WinWin artifacts and relationships, including the basic negotiation model discussed earlier.

Focus on ease of use to foster stakeholder involvement. People participating in a requirements negotiation typically don't have time to take training before starting to negotiate. Ease of use lets more people directly participate and elicits more from everybody involved. This leads to better buy-in because more interests can be accommodated earlier in the process. It also helps to develop broader and deeper deliverables. EasyWinWin results surpass 3G WinWin results in terms of the number of artifacts collected in a negotiation.[13]

In addition, the higher number of issues identified and resolved helps reduce risks early in a project and the chances of it derailing later. For example, our 3G WinWin digital library requirements negotiations involved 15 to 25 win conditions and converged in two to three weeks (as compared to two to three months for comparable manual win–win requirements negotiations). Similar EasyWinWin requirements negotiations involved 50 to 120 win conditions and converged in two to three days. Furthermore, we found that the stakeholders' experience with EasyWinWin led to better mutual understanding and greater stability of the negotiated requirements.

Provide a robust infrastructure. The 3G homemade database server was very fragile and prone to losing people's work in crashes. Using a reliable infrastructure is critical to avoid frustration and to ensure stakeholder buy-in.

Provide interoperability. 3G WinWin did not interoperate well with other groupware systems, even after we built an applications programming interface. One industry project successfully built and applied its own WinWin overlay on top of the groupware system it was using but did not try to develop a more general capability.

Support for multiple modes of interaction. Beyond same-time and same-place stakeholder interaction, we successfully included remote

Using a reliable infrastructure is critical to avoid frustration and to ensure stakeholder buy-in.

You need to consider how well the user base for which you are building the groupware system represents mainstream end users.

participants in EasyWinWin workshops by using the Web-based capabilities of GroupSystems.com and audio links. We are currently expanding recommendations for geographically distributed teams intending to adopt EasyWinWin activities in different time and place settings.

WinWin project lessons

Finally, here are some lessons we learned in project organization, user involvement, and expectations management.

Involve mainstream end users. With persistence, and by focusing on your mainstream end users, you can develop groupware systems that both speed up the initial definition process and help stakeholders achieve a shared vision with lasting value across the application's entire life cycle. Thus, you need to consider how well the user base for which you are building the groupware system represents mainstream end users. Once we had an annual set of USC projects to support with 3G WinWin, we overfocused on USC users rather than on our primary target of mainstream industry users.

When developing groupware, perseverance pays off. Do not overreact to initial negative experiences. Groupware systems must be carefully balanced to accommodate the many stakeholders' different needs. When designing 2G and 3G WinWin, we reacted to the 1G WinWin experience with its high-priced commercial infrastructure by building a homemade infrastructure. The EasyWinWin overlay above GroupSystems.com's infrastructure has been much more successful. When designing 3G WinWin, we also overreacted to some instances of artifact misuse in 2G WinWin by creating a system whose rules were so rigorous that they turned off most users. 3G WinWin improved on 2G WinWin with its well-defined architectural interfaces but lost out because of its inflexibility for mainstream stakeholder groups. Relative to the "build it twice" guidance in Winston Royce's initial waterfall model article[14] and in Fred Brooks' *The Mythical Man Month*,[15] you must also add Brooks' *second system syndrome*: Developers, particularly for groupware, are likely to react overambitiously to experiences with initial prototypes or systems.

Use the system to plan its own future. Doing this provides both a good test of the current groupware system and a good way of achieving a shared vision of its future directions. Both USC's experience with using 1G WinWin to negotiate requirements for 2G WinWin and GroupSystems.com's similar experience with using EasyWinWin substantiates this.

GSS developers should not expect to get the system right the first time—or even the second time. Our experience with the four generations of WinWin requirements negotiation systems is that it takes several iterations of operational GSSs to fully realize their benefits. Even now, we are involved in a fifth iteration to provide better support for less experienced facilitators. However, the payoffs are worth it: We have experienced about a factor of four improvement in multistakeholder requirements negotiation time when going from manual negotiation to the 2G to 3G WinWin system, and another factor of five in going from 2G-3G WinWin to the EasyWinWin system. In addition, the negotiation results have become more thorough and better internalized by the stakeholders.

Acknowledgments

DARPA, through Rome Laboratory under contract number F30602-94-C-0195, and the Austrian Science Fund, with an Erwin Schrödinger research grant for Paul Grünbacher (1999/J 1764), sponsored this research. It was also sponsored by the affiliates of the USC Center for Software Engineering: Aerospace, Automobile Club of Southern California, Boeing, C-Bridge, Chung–Ang University (Korea), Draper Labs, Electronic Data Systems Corporation, Federal Aviation Administration, Fidelity, GDE Systems, GroupSystems.com, Hughes, Institute for Defense Analysis, Litton Industries, Lockheed Martin Corporation, Lucent Technologies, Microsoft, Motorola, Northrop Grumman Corporation, Rational Software Corporation, Raytheon, Science Applications International Corporation, Software Engineering Institute (Carnegie-Mellon University), Software Productivity Consortium, Sun Microsystems, Telcordia Technologies, TRW, US Air Force Research Laboratory, US Army Research Laboratory, US Army TACOM, and Xerox Corporation.

We also thank the definers and developers of the first three versions of WinWin: Ellis Horowitz, Dan Port, Prasanta Bose, Yimin Bao, Anne Curran, Alex Egyed, Hoh In, Joo Lee, June Lee, Mingjune Lee, and Jungwon Park. We also thank the users of the four WinWin systems: Frank Beltz, Garry Brannum, Walter

Green, Elizabeth Kean, Judy Kerner, Julie Kwan, Andrew Landisman, Anne Lynch, Ray Madachy, Azad Madni, Nikunj Mehta, Steve Mosher, Karen Owens, Arnold Pittler, Michael Saboe, and John Salasin.

References

1. The Standish Group, *CHAOS Report*, 1995, www.standishgroup.com/visitor/chaos.htm (current 16 Apr. 2001).
2. D. Waitley, *The Double Win*, Berkeley Books, New York, 1985.
3. S. Covey, *The Seven Habits of Highly Effective People*, Fireside Books, New York, 1990.
4. B. Boehm and R. Ross, "Theory W Software Project Management: Principles and Examples," *IEEE Trans. Software Eng.*, July 1989, pp. 902–916.
5. B. Boehm et al., "Software Requirements as Negotiated Win Conditions," *Proc. Int'l Conf. Requirements Eng.*, IEEE Press, Piscataway, N.J., 1994.
6. B. Boehm, "Requirements That Handle IKIWISI, COTS, and Rapid Change," *Computer*, July 2000, pp. 99–102.
7. J. Highsmith, *Adaptive Software Development*, Dorset House, New York, 2000.
8. J. Nunamaker et al., "Lessons from a Dozen Years of Group Support Systems Research: A Discussion of Lab and Field Findings," *J. Management Information Systems*, vol. 13, no. 3, Winter 1996–1997, pp. 163–207.
9. P. Gruenbacher, "Collaborative Requirements Negotiation with EasyWinWin," *Second Int'l Workshop Requirements Eng. Process*, IEEE CS Press, Los Alamitos, Calif., 2000.
10. *The EasyWinWin Process Guide: USC-CSE and GroupSystems.com*, http://sunset.usc.edu/research/WINWIN, 2000 (current 16 Apr. 2001).
11. J.B. Harvey, *The Abilene Paradox and Other Meditations on Management*, Jossey-Bass, San Francisco, 1988.
12. B. Boehm et al., "Using the WinWin Spiral Model: A Case Study," *Computer*, 1998, pp. 33–44.
13. A.F. Egyed and B. Boehm, "Comparing Software System Requirements Negotiation Patterns, *J. Systems Eng.*, vol. 2, no. 1, 1999.
14. W.W. Royce, "Managing the Development of Large Software Systems," *Proc. IEEE WESCON*, IEEE Press, Piscataway, N.J., 1970, pp. 1–9.
15. F.P. Brooks, *The Mythical Man Month*, Addison-Wesley, Reading, Mass, 1975.

About the Authors

Barry Boehm is TRW Professor of Software Engineering and director of the Center for Software Engineering at the University of Southern California. His current research focuses on integrating a software system's process models, product models, property models, and success models via an approach called MBASE (Model-Based Architecting and Software Engineering). He received his BA from Harvard and his MS and PhD from UCLA, all in mathematics. He received an honorary ScD in computer science from the University of Massachusetts. He is an AIAA fellow, an IEEE fellow, an INCOSE fellow, an ACM fellow, and a member of the National Academy of Engineering. Contact him at the USC Center for Software Engineering, Los Angeles, CA 90089-0781; boehm@sunset.usc.edu.

Paul Grünbacher is an assistant professor of systems engineering and automation at the Johannes Kepler University Linz, Austria. His research interests include the application of collaborative technology in software engineering with a focus on requirements engineering and methodologies for software process improvement. He studied Business Informatics and holds a PhD from the University of Linz. Contact him at Systems Engineering and Automation, Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria; pg@sea.uni-linz.ac.at.

Robert O. Briggs is director of methodology and process tools for GroupSystems.com, where he oversees the future evolution of GroupSystems software, and is also research coordinator at the Center for the Management of Information at the University of Arizona. He investigates the cognitive foundations of collaboration with a focus on the development and deployment of software and processes to enhance the performance of teams making a joint effort toward a goal. He earned his BS in Information Systems and Art History and an MBA from San Diego State University. He earned his PhD in management and information systems from the University of Arizona. Contact him at 1430 E. Fort Lowell Rd. #301, Tucson, AZ 85719; bbriggs@groupsystems.com.