

2009

PACIFIC NW SOFTWARE  
QUALITY  
CONFERENCE



MOVING  
QUALITY  
FORWARD

OCTOBER 27-28, 2009

Conference Paper Excerpt

From the

CONFERENCE  
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

# A Distributed Requirements Collaboration Process

Brandon.Rydell@gmail.com  
Sean.D.Eby@gmail.com  
Carl.Seaton@gmail.com

## Abstract

Distributed project teams struggle to accurately document the negotiations and trade-offs required to lead teams to concise, prioritized requirements. The main reasons for this challenge are 1) a lack of objective criteria to evaluate requirements and 2) a lack of freely available tools to channel negotiations into objectively characterized requirements which can be comparatively evaluated.

While there are numerous publications and tools addressing the challenges of software requirements engineering, many organizations today still manage requirements as tabular lists. Individual requirements are often modified during negotiations between groups such as engineering, marketing and management. Frequently much of this negotiation takes place informally in numerous face to face meetings where much of the rationale for changes and trade-offs are not documented and are subsequently lost.

In this paper we outline a new requirements collaboration process to address these issues. This new process is based on work done by Karl Wiegers in his paper [Prioritizing Requirements](#) [1] and is extended to include a method of proposing alternate requirements, documenting the negotiation of priority among interested parties and a way to rationally select priority requirements based on an objective measure of their relative merit. We also introduce an application prototype called the Distributed Requirements Collaboration Tool (DRCT) that was built for use by distributed teams to support the requirements negotiation process described above and also to address the issue of capturing rationale as requirements are negotiated. Lastly, we will discuss our experience using the DRCT to identify priority requirements for a more functional version of the DRCT.

## Biography

Brandon Rydell is a software engineering supervisor at Portland General Electric (PGE) in Portland, Oregon. His experience over the last two decades includes working as a software requirements engineer and project manager (PM) on a wide variety of software engineering projects for PGE, PacifiCorp, Nike and United Parcel Service. Brandon has a Masters of Software Engineering (MSE) degree from Portland State University, and a B.A. in Business Administration from the University of Washington. He is also a certified Project Management Professional (PMP).

Sean Eby is currently a software engineer at PGE in Portland, Oregon. In his 11+ years of experience he has worked in all areas of software engineering for companies such as: Coaxis, Emery Worldwide, Merant and NCD. Sean specializes in designing and developing simple, elegant software solutions. Sean has an MSE degree from Portland State University.

Carl Seaton is a Staff Systems Software Engineer at Arris, currently developing high-performance video-on-demand servers at the Beaverton, Oregon site. Over the past 16 years Carl has done everything from customer support to software development to engineering management, specializing in highly reliable distributed systems. Carl holds B.S degrees in Computer Engineering and Computer Science from Oregon State University, and will complete his Master of Software Engineering degree at Portland State University this Fall.

Copyright: Brandon Rydell, Sean Eby and Carl Seaton August 1<sup>st</sup> 2009

## Introduction

Fred Brooks said, "The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." [2]

Requirements elicitation, specification and management are the fundamental tools used by software engineers when deciding precisely what to build. A common challenge with requirements management for organizations is to accurately document the negotiations and trade-offs that are required to lead teams to concise prioritized requirements, especially within distributed teams.

Two main reasons for this challenge include:

1. A lack of objective criteria to evaluate requirements (frequently requirements are dictated by vertical teams like marketing / engineering, etc.. and presented as simple lists) and
2. A lack of freely available tools to channel negotiations into objectively characterized requirements which can be comparatively evaluated.

While there are countless books, papers and websites addressing the challenges of software requirements engineering, and there are even commercial requirements management tools available, many organizations today still use basic documents or spreadsheets to manage simple requirements lists. This is due to a lack of software requirements engineering knowledge, skill and application.

In some more advanced organizations, requirements are often modified during negotiations between engineering (technical feasibility and cost), marketing (market benefit), and management (schedule, resources and strategic business goals). Frequently much of this negotiation takes place informally face to face during meetings where much of the rationale for alternatives, changes and trade-offs are not recorded. This challenge becomes increasingly difficult with physically and / or temporally distributed teams.

In this paper we outline a new requirements collaboration process which addresses these issues and as a result offers a process that is more suited to the needs of distributed teams. The development of this new process began by using Karl Wiegers' requirements prioritization process as a base. We then extended it to include a method of proposing alternate requirements, negotiating the priority of requirements among interested parties, capturing the trade-offs made along the way and rationally selecting priority requirements based on their relative rating on item (ROI) scores. See the [Wiegers Prioritizing Requirements](#) paper for a more complete description of a semi-quantitative approach to requirements prioritization, which we then adapted to develop our ROI concept.

We also introduce an application prototype called the Distributed Requirements Collaboration Tool (DRCT) built for use by distributed teams which supports the requirements negotiation process described above and addresses the issue of capturing rationale as requirements are negotiated.

## Distributed Requirements Collaboration

We believe requirements elicitation, specification and management as a software engineering topic is of particular value to study for two main reasons:

1. The implications of Brooks' compelling statement above, made so many years ago, which in our experience holds true still today.
2. In our experience and training, the application of the discipline of software requirements engineering has been shown to have a good return on investment. Research by Boehm confirms this [3], yet software requirements engineering today remains insufficiently applied.

## I. New Distributed Requirements Collaboration Process

When deciding whether it was necessary to develop a new distributed requirement collaboration process to meet the needs of distributed teams and to address the challenges given above we first reviewed two papers on the topic.

In Prioritizing Requirements Karl Wiegers describes a relatively simple process of rating and ranking requirements that is effective enough in our view, practical, and easy to use but does not explicitly extend well to distributed teams nor does it include support for capturing the rationale for trade-offs that frequently occur as requirements are negotiated and prioritized.

The paper titled Supporting Distributed Collaborative Prioritization for WinWin Requirements Capture and Negotiation [4] discusses a tool that was developed to support collaborating teams with prioritizing requirements in a distributed setting using the WinWin development model developed by B. Boehm. The tool includes support for multiple stakeholders, capturing rationale, and gives a much richer way to visualize the complexity of requirements prioritization. After review, our team decided that the resulting tool and process was more complex than we needed.

As a result, we developed the process summarized in the outline below (and also illustrated in Figure 1) which suits our specific needs. It is a simple process like the one introduced by Wiegers but extended to distributed teams and includes support for capturing rationale for trade-offs when negotiating the priority of requirements that are typical to projects we encounter.

### Process Outline:

A. Propose a list of requirements that need prioritization. Note that requirements that are non-negotiable and must be included in a given project need not be prioritized and can simply be assumed part of the project.

For each proposed requirement:

1. Provide a name, description and rationale.
2. Assign (person in marketing role) a benefit and a penalty rating (based on a scale of 1 – 9, where 1 indicates low benefit or low penalty avoidance and 9 equals high benefit or high penalty avoidance) including rationale.
3. Assign (person in engineering role) a cost and risk rating (based on a scale of 1 – 9, where 1 indicates low cost or low risk and 9 equals high cost or high risk) including rationale.
4. Calculate an ROI.  $ROI = (benefit + penalty) \div (risk + cost)$ . The system performs this calculation based on ratings given.
5. Optionally, propose an alternate requirement and repeat steps 2 – 5 above.

B. Select the final list of requirements from the proposals negotiated above for inclusion in the cut list (person in the PM role).

Process Flow:

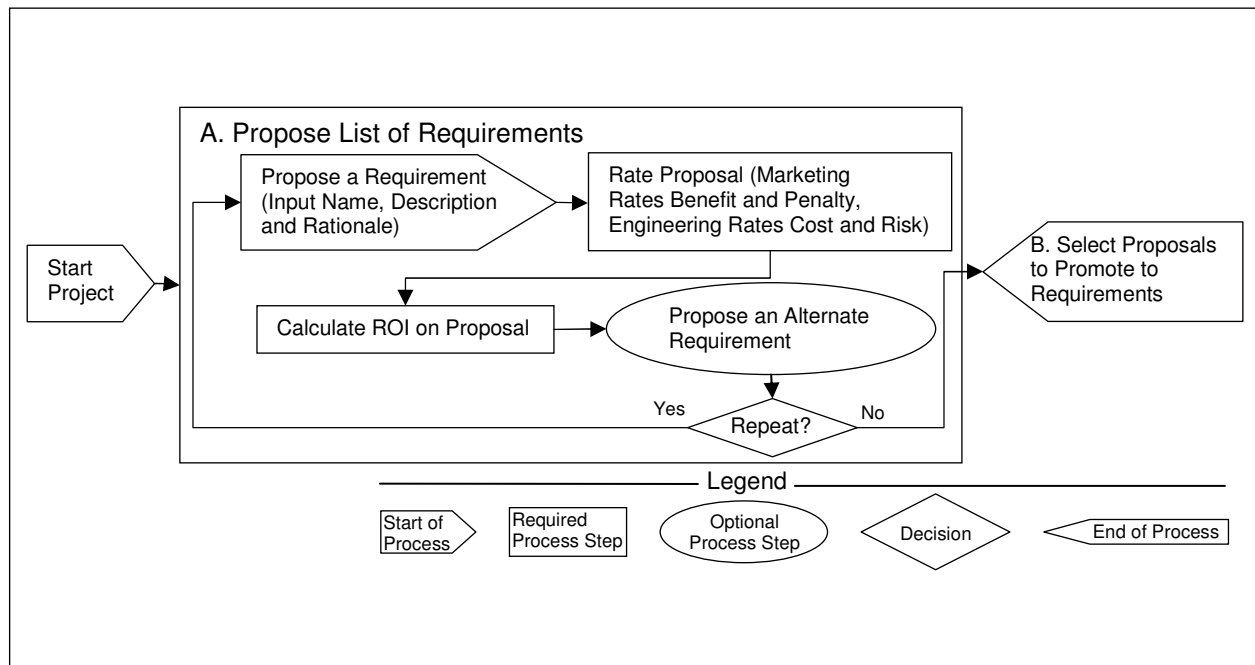


Figure 1 – Requirements Proposal and Negotiation Process

This process is effectively distributed by extending it to all project team members via an internet or intranet page (sample pages are provided in the following section).

From the process above, requirements are proposed, alternatives may be included, priority is effectively negotiated through ratings given by the respective roles, and because it is an application, a history of events including rationale is easily captured for the sake of documentation.

## II. Distributed Requirements Collaboration Tool (Application Prototype)

The prototype we developed to support the process above is designed to perform the following functions:

1. Authenticate User
2. View Project(s) - projects are used to organize proposed requirements
3. Create Project(s)
4. Create Proposal(s)
5. View / Rate Proposal(s)
6. Propose Alternative(s) - alternatives are simply another independent proposal
7. Promote Proposal(s) to Requirement(s)
8. View Requirements

Note that in steps 4 – 6, the rationale for each can be recorded in the DRCT.

The sample wireframe screens below are taken from the actual prototype we developed and deployed to the internet. These screens are provided below in the same order as the processes listed above. Both the Authenticate User and Create Project(s) steps are not illustrated due to their simplicity and to preserve brevity.

Note that we used the colors and symbols below to indicate the following in the prototype:

- Grey and ■ - Current user's role (PM, marketing or engineering) does not allow editing.
- Red and ► - Rating is incomplete and needs to be filled in by another role or in the case of the ROI field it means that one or more of the ratings required by the current role needs to be completed before the system can calculate ROI.
- Green and ▼ - Rating is incomplete and needs to be filled in by current user, or in the case of the ROI field, it means that all the permissible inputs for the role currently logged in are complete.
- White and ● - Rating is complete.

1. View Project(s) – Upon logging in, the user will be taken to the “Projects” page. All projects are viewable here. No projects will be shown if none have been created. Note that multiple projects can be created by clicking the “New project” button shown at the bottom right. In this example we assume that a PM views or creates projects.

<b>Distributed Requirements Collaboration Tool</b>	
Welcome brandon!	Role: Project Manager   <a href="#">Logout brandon</a>
<b>Projects:</b>	
Name	Description
<a href="#">DRCT 1.0 Requirements</a>	Project to collect and prioritize the non-basic requirements for the Distributed Requirements Collaboration Tool version 1.0.
<a href="#">Proto Testing</a>	Proto Testing
<a href="#">Text Editor</a>	Text
<input type="button" value="New project"/>	

2. Once a project has been created, it can be edited. More importantly, new proposals can be added to it. New proposals are added by clicking the “New proposal” button at the bottom right of this screen. In this example we assume that the PM creates a new proposal (shown next).

<b>Distributed Requirements Collaboration Tool</b>		
Role: Project Manager   <a href="#">Logout brandon</a>		
<b>Editing Project: "Text Editor"</b>		
Name:	<input type="text" value="Text Editor"/>	
Description:	<input type="text" value="Text"/>	
<input type="button" value="Cancel"/> <input type="button" value="Save"/>		
<b>Requirements:</b>		
Name	Description	ROI
<b>Proposals:</b>		
Name	Description	ROI
<input type="button" value="New proposal"/>		

3. Create Proposal(s) – This screen is used to create new proposed requirements.

## Distributed Requirements Collaboration Tool

---

Role: Project Manager | [Logout brandon](#)

### Creating Proposal

Name:	Unlimited Undo
Description:	Undo all actions
Rationale:	No one else does this

$$\left( \text{Benefit: } \blacksquare + \text{Penalty: } \blacksquare \right) / \left( \text{Cost: } \blacksquare + \text{Risk: } \blacksquare \right) = \text{ROI: } \blacksquare$$

3a. Once saved, new proposals appear on the “Editing Project” screen under the “Proposals.”

## Distributed Requirements Collaboration Tool

---

Proposal was successfully created.

Role: Project Manager | [Logout brandon](#)

### Editing Project: "Text Editor"

Name:	Text Editor
Description:	Text

#### Requirements:

Name	Description	ROI

#### Proposals:

Name	Description	ROI
Unlimited Undo	Undo all actions	▶ <input type="button" value="Destroy"/>

4. Proposals may be viewed and rated by selecting them by name from the "Editing Project" screen (shown previously). Here we see that an engineer has rated the "Cost" and "Risk" of the proposal.

## Distributed Requirements Collaboration Tool

Proposal was successfully updated. Role: Engineering | [Logout sean](#)

**Editing Proposal: "Unlimited Undo"** [Back to Proposals](#)

Name:	Unlimited Undo
Description:	Undo all actions
Rationale:	No one else does this

$$\left( \text{Benefit: } \blacksquare + \text{Penalty: } \blacksquare \right) / \left( \text{Cost: } 9 + \text{Risk: } 9 \right) = \text{ROI: } \blacksquare$$

**Alternate Proposals:**

Name	Description	ROI
<u>Unlimited Undo</u>	Undo all actions	▶ <input type="text"/>

4a. Here, marketing has provided ratings for "Benefit" and "Penalty" on the same proposal.

## Distributed Requirements Collaboration Tool

Proposal was successfully updated. Role: Marketing | [Logout carl](#)

**Editing Proposal: "Unlimited Undo"** [Back to Proposals](#)

Name:	Unlimited Undo
Description:	Undo all actions
Rationale:	No one else does this

$$\left( \text{Benefit: } 9 + \text{Penalty: } 9 \right) / \left( \text{Cost: } 9 + \text{Risk: } 9 \right) = \text{ROI: } 1.0$$

**Alternate Proposals:**

Name	Description	ROI
<u>Unlimited Undo</u>	Undo all actions	1.0
<u>Single Undo</u>	Undo last actions	▼ <input type="text"/>

5. Propose Alternative(s) – The “Creating Alternate for Proposal” screen is accessed by clicking the “Alternate” button on the previously pictured screen. Here, engineering proposes an alternative to the original proposal, which they suggest (in the “Rationale” field), will be easier to implement.

## Distributed Requirements Collaboration Tool

---

Role: Engineering | [Logout sean](#)

---

**Creating Alternate for Proposal: Unlimited Undo**

Name:	Single Undo
Description:	Undo last actions
Rationale:	Much easier than undoing all actions

$$\left( \text{Benefit: } \uparrow + \text{Penalty: } \uparrow \right) / \left( \text{Cost: } \downarrow + \text{Risk: } \downarrow \right) = \text{ROI: } \uparrow$$

5a. Marketing shown here enters an alternate proposal of their own and inputs their respective rating values and rationale as before.

## Distributed Requirements Collaboration Tool

---

Role: Marketing | [Logout carl](#)

---

**Creating Alternate for Proposal: Single Undo**

Name:	Limited Undo
Description:	Undo last 5 actions
Rationale:	Easier than undoing all actions, more functional than only undoing the last action

$$\left( \text{Benefit: } \downarrow + \text{Penalty: } \downarrow \right) / \left( \text{Cost: } \uparrow + \text{Risk: } \uparrow \right) = \text{ROI: } \uparrow$$

The negotiations continue until a list of mutually rated proposals and / or alternates is created. In the examples above, proposals and / or alternates once rated by marketing, will then be rated by engineering and vice versa.

6. The PM then, based on calculated ROI, evaluates each of the negotiated proposals and / or alternatives, and then selects those which he wants to promote by clicking the “Promote” button. In this example the PM is viewing the “Limited Undo” proposal just prior to promoting it.

## Distributed Requirements Collaboration Tool

---

Role: Project Manager | [Logout brandon](#)

**Editing Proposal: "Limited Undo"**

[Back to Proposals](#)

<b>Name:</b>	Limited Undo
<b>Description:</b>	Undo last 5 actions
<b>Rationale:</b>	Easier than undoing all actions, more functional than only undoing the last action

$$\left( \text{Benefit: } \boxed{7} + \text{Penalty: } \boxed{7} \right) / \left( \text{Cost: } \boxed{4} + \text{Risk: } \boxed{4} \right) = \text{ROI: } \boxed{1.8}$$

---

**Alternate Proposals:**

Name	Description	ROI
<u>Limited Undo</u>	Undo last 5 actions	1.8
<u>Unlimited Undo</u>	Undo all actions	1.0
<u>Single Undo</u>	Undo last actions	0.5

Note that the rationale provided during the negotiation process can also be used as a consideration by the PM when deciding which proposal / alternative(s) to promote.

7. View Requirements – Finally, any team member can see all the proposals that have been promoted to requirements, as shown here. Note the option to “Destroy” proposals is present for use by the PM.

## Distributed Requirements Collaboration Tool

---

Proposal was successfully promoted. Role: Project Manager | [Logout](#) [brandon](#)

---

**Editing Project: "Text Editor"**

Name:

Description:

**Requirements:**

Name	Description	ROI
<u>Limited Undo</u>	Undo last 5 actions	1.8

**Proposals:**

Name	Description	ROI
<u>Unlimited Undo</u>	Undo all actions	1.0 <a href="#">Destroy</a>
<u>Single Undo</u>	Undo last actions	0.5 <a href="#">Destroy</a>

This entire process is iterative and can produce lists of objectively prioritized requirements of unlimited length. As such lists are built within the DRCT by following this process, a comprehensive trail of documentation is produced as a valuable byproduct.

### III. Evaluation of the DRC Process and DRCT

Here we describe our experiences with using the DRC process and tool, and present our conclusions on the effectiveness of using them.

#### Wiegiers' Process

Using the Wiegiers paper as a base to build our process off of worked well. Using subjective scoring made it easy to understand, while the objective ROI provided an effective tool to support decision making. This simple, effective approach made for a solid foundation to build the DRCT on.

One of the early challenges we faced was consistently applying rating values to each of the proposed requirements. While the rating scores are subjective individually, they must be consistent relative to each other in order to produce usable ROI scores. The DRCT left this challenge to the operators but some automated mechanism to support this would likely make sense as a new feature.

We found the Wiegiers recommendation from the paper, to keep the requirements evaluated at a high level, to be true. We initially put in very detailed requirements, but found that the volume of data generated made it more difficult to manage all the proposals together. To overcome this, we grouped the detailed proposals together into higher level proposals. This increased the effectiveness of the process.

## Using the Prototype

Using the prototype to evaluate the process gave us good insight into what did and did not work well. Doing so also gave us good feedback on what needed to change about the DRCT in order to make it more usable, plus it helped us define more valid requirements in the process.

Because the DRCT as a prototype was immature, we found several areas that inhibited our use of it, and therefore, the evaluation of the new process. One case in particular was the lack of rationale per rating value. This made it more difficult to understand why another team member scored things in certain ways, and that lack of communication made it harder to respond with alternatives that addressed those unspoken issues. We used email at first to overcome this but found it to be somewhat ineffective due to lack of context.

Because we built the DRCT as part of our MSE practicum, we had to severely curtail the scope of the DRCT to stay on schedule. This gave us time to complete the DRCT and use it for the evaluation, but the resulting simplicity of the DRCT made it less effective to evaluate the new process and tool under real-world conditions. Again, due to this simplicity, we did not need to spend as much time during negotiations because of the relative simplicity of the proposed feature set, and the lack of more complex real-world tradeoffs limited the insight we could gain. However, the initial signs were encouraging.

The negotiation process was effective in promoting dialogue between participants. It forced stakeholders to communicate and clarify requirements, rationale, benefit and cost, ultimately leading to better decisions supporting the selection of final requirements for a cut list. We found that capturing those discussions to serve as documentation behind the decision-making process was especially useful.

The promotion step within this process was also effective. Explicit promotion was beneficial by having a third party (the PM) determine when negotiations would conclude. This helped to maintain the project's momentum. Having an independent role choose the requirements to promote also gave more objectivity to the process. While engineering and marketing debated, each had to keep in mind that a third party would make the choice based on the persuasiveness of the arguments.

One of the more interesting discoveries was in the ability for the PM to override the ROI calculation and simply choose to promote a proposal to a requirement for more subjective reasons. For example, an error-handling proposal had the lowest ROI value, but the PM thought, and we agreed, that it needed to make the cut list regardless of the ROI. The ability of the Project Manager to override the ROI is a useful correction mechanism.

As a result of the process and DRCT evaluation, we think the new distributed process that we practiced using the DRCT was practical and would be applicable to building commercial software professionally. We ended up with a working prototype that supported negotiating requirements among team members who are physically and temporally separated. From this prototype we developed an excellent understanding of what features the first release of the new tool should have in order to be a powerful and effective requirements elicitation and management tool. We achieved mutual agreement on our cut list (discussed below) as a reasonable and valid set of priority requirements for the next iteration of the DRCT.

We would be interested in seeing the DRCT and process used on a more complex project in order to validate them further. Our initial experience with them suggests that they would be useful on small to mid-sized projects where prioritizing requirements among distributed participants is needed.

## DRCT 1.0 Requirements Cut List

As mentioned above, the authors used the DRCT to negotiate a proposed new set of requirements for a next version of the DRCT.

Once each proposal had a negotiated ROI, a cut list of requirements was selected by the project manager for the next release of the project. Using the calculated ROI values, rationale and some judgment, it was easy to select the requirements which made the most sense functionally, and which were agreed to have the greatest immediate value to the product. This process of selecting a short list was led by the project manager and offered a powerful check on negotiations that could have otherwise run on indefinitely.

A sample from this negotiated cut list is included below as Figure 2.

Name	Description	Rationale	Benefit	Penalty	Cost	Risk	ROI
Create users	Enable creation of user logins which can be assigned to individual stakeholder groups.	People will need to be able to use the tool, and having an admin modify the database for this would result in unnecessary delays.	4	9	2	2	3.3
Project audit trail	Should have an audit trail for projects, requirements and proposals (who, what, when, why).	Tracking changes allows users to more easily reevaluate things in the future. It also allows for easier tamper detection and can help correct user errors.	9	5	5	3	1.8
Provide rationale for each rating value	Rationale should be collected for each rating value, as well as for the proposal itself. Rationale should be updated when the rating value changes.	Rationales are necessary so that the viewers know the meaning behind the rating value given. This enhances discussions and decisions in the future as conditions change.	9	3	4	3	1.7
Encrypted passwords	Passwords should not be sent or stored in the clear.	This is an easy security hole to exploit and would not be acceptable to many organizations.	1	7	4	3	1.1
Error handling + logging	Display friendly error to user, log error details in database.	Users should be notified of an error occurring, but the tool should also log the error to assist technical support and engineering.	4	7	6	4	1.0

Figure 2 – DRCT V 1.0 Negotiated Cut List

## Conclusion

While accurately documenting the negotiations and trade-offs required to lead teams to concise, prioritized requirements is often a challenge for distributed project teams, it can be achieved with a process and tool fit for the job. Making this process and tool fit to the task of managing requirement negotiations is worth undertaking, as it will invariably save time and money on most software projects.

In this paper we have introduced a new simple requirements collaboration process and tool for use by distributed teams that we believe will make managing requirements more effective for them. This new process and tool includes a method and means of proposing requirements, negotiating alternatives, collaboratively rating the proposals to determine their relative priority and then rationally selecting a requirements cut list from all the proposals offered. Distributed teams using this process and tool will enjoy the benefit of not only completing this work remotely but also capturing the rationale for making some proposals into requirements as they negotiate each proposal's relative merit.

Our experience with using this new process and tool in a distributed setting was positive. We think this new process and tool would be useful on many of the software projects on which we typically work.

## References

- [1] – Wiegers, K. 1999. First Things First: Prioritizing Requirements - <http://www.processimpact.com/articles/prioritizing.html>
- [2] – Brooks, F. 1986. No Silver Bullet, Essence and Accidents of Software Engineering
- [3] – Boehm, B. 1981. Software Engineering Economics
- [4] – Park, J-W. 1999. Supporting Distributed Collaborative Prioritization for WinWin Requirements Capture and Negotiation - <http://csse.usc.edu/csse/TECHRPTS/1999/usccse99-516/usccse99-516.pdf>

## Acknowledgements

The authors would like to acknowledge and offer our gratitude to the faculty, staff and students at the Portland State University Oregon Masters of Software Engineering (OMSE) program, including Dr. Kalman (Kal) Toth, Dr. Stuart Faulk, professor Manny Gatlin and the rest of the instructors, staff and students supporting this program. Additionally, our thanks go to our wives and children, whose patience and support of us during the development of this project are truly appreciated. A final thank you goes to Carol Oliver of Stanford University and Ganesh Prabhala, who have offered their time and patience to review this work.