

Using a Hybrid Method for Formalizing Informal Stakeholder Requirements Inputs

Hasan Kitapci, Barry W. Boehm
University of Southern California
Center for Software Engineering
90089 Los Angeles, CA
{hkitapci, boehm}@cse.usc.edu

Abstract

Success of software development depends on the quality of the requirements specification. Moreover, good – sufficiently complete, consistent, traceable, and testable – requirements are a prerequisite for later activities of the development project. Without understanding what the stakeholders really want and need, and writing these requirements, projects will not develop what the stakeholders wanted.

During the development of the WinWin negotiation model and the EasyWinWin requirements negotiation method, we have gained considerable experience in capturing informal requirements in over 100 projects. However, the transition from informal representations to semi-formal and formal representations is still a challenging problem.

Based on our analysis of the projects to date, we have developed an integrated set of gap-bridging methods as a hybrid method to formalize informal stakeholder requirements inputs. The basic idea is that orchestrating these gap-bridging methods through the requirements engineering process can significantly eliminate requirements related problems and ease the process of formality transition.

1. Introduction

Requirements engineering is a crucial first step in the software development process. Whether you are building custom systems or systems by assembling components, using commercial-of-the-shelf (COTS) software or making changes to existing software, you still need to explore, capture, and communicate the requirements.

A critical success-factor in requirements elicitation and negotiation is the availability and participation of

key stakeholders. However, stakeholders who can contribute most value, insights, and realism often have the least time to participate in such a process. In such cases, one would like to use their limited time to participate in defining and negotiating among the most critical requirements for the system. Therefore an effective and efficient way to elicit, document, and validate requirements is crucial.

Over the years much work has been done to improve the productivity and quality of the early stage of software life cycle. However, requirements engineering still remains as a hard problem to automate due to its inherently human-centered nature. In his article [36], Karl Wiegers indicates that the most essential and yet often neglected practice is to write down, or document, the requirements. Failures to document requirements in sufficiently complete, consistent, traceable, and testable format is a serious risk to project success. Moreover, documents containing defects that used as references in further processes will cause more defects in the product's design, code, and documentation. These defects in requirements cause many to fail or overrun their schedule or budget [35]. Boehm [3] has reported that although only 6 percent of the project cost and between 9 and 12 percent of project duration is spent in the requirements phase, it costs between five to ten times more to repair errors during coding than during the requirements phase: and it costs between 100 and 200 times more during maintenance.

During the development of the WinWin negotiation model [4] and the EasyWinWin requirements negotiation method [6] we have gained considerable experience in capturing informal requirements in over 100 projects. EasyWinWin helps a team to jointly brainstorm, organize, and negotiate informal inputs into a comprehensive set of artifacts about the goals and preferences of stakeholders, existing conflicts, and

achieved agreements. Nevertheless, the result is not a complete, consistent, traceable, and testable requirements specification. During our software engineering course, students formed as teams experienced using EasyWinWin negotiation results with the help of Model-Based Architecting and System/Software Engineering (MBASE) requirements template [7] to formalize the informal requirements to more precise requirements specification.

Our lessons learned during real-world negotiations show that it is not effective to dampen the enthusiasm of stakeholders and the creative flow of ideas with techniques emphasizing detailed wordsmithing, correctness, consistency, etc. [6]. However, post-processing and improving informal inputs into more precise and realistic requirements for the stakeholders to review and iterate is not well understood. A failure to successfully bridge the gap from rather informal inputs to more formal representations may have serious negative impacts on contract negotiation, initial project planning, architectural decisions, or end-item deliverables [17].

There is a need to develop an integrated environment that brings together techniques and solutions to the issues of requirements engineering process where all success-critical stakeholders can participate in documentation and validation of requirements. Thus, we have been exploring alternative methods for bridging the gap between informal inputs and a requirements specification: template-based refinement, natural language processing, keyword analysis, formal methods expertise, and inspections. The main focus of this research is orchestrating those methods in order to reduce the number of defects in requirements specification and assist in documenting and analyzing the requirements specifications.

The remainder of this paper is organized as follows: Section 2 compares relative strengths of various target requirements representation methods. Section 3 describes the analysis of different approaches for formalizing informal requirements. Section 4 explains our hybrid method from gap-bridging approaches. Section 5 discusses our software development process, the EasyWinWin negotiation approach for acquiring stakeholder inputs, hybrid method applied to e-services projects, and discusses typical characteristics of results. Section 6 discusses the related works. Conclusions and an outlook on further work summarize progress to date and prospects for the future.

2. Degree of formality

Developing software is an evolutionary process that adopts informal, semi-formal, and formal representations of knowledge. Managing the transitions between these representations is a key to the success of any engineering task. On the one hand informal representations for capturing system requirements support straightforward involvement and interaction of even inexperienced stakeholders. On the other hand this approach may result in inconsistent, incomplete, and ambiguous specifications. Semi-formal and formal representations overcome these problems but are less suited for stakeholder interaction.

Our analysis of negotiation results concludes that an approach supporting the formalization of informal EasyWinWin inputs should address both quantity and quality aspects: we need (1) scaleable techniques addressing the size and complexity of negotiation results and expediting the refinement into a more formal requirements specifications, and (2) techniques helping to identify and remove defects from the negotiation results early on while product complexity is still manageable. Our first challenge was to determine an appropriate degree of formality to serve as the initial target for methods of strengthening informal requirement inputs. In surveying requirement practices among USC (University of Southern California)'s industry affiliates, we found four primary categories of formality: formal specifications, formatted specifications, itemized statements, and stories. Each of these methods has been found useful to some constituencies. The results of our comparison of the relative strengths of them showed that the strongest method was formatted specifications [23]. For this reason, we chose it as the initial target to use in strengthening informal requirements inputs.

3. Gap-bridging approaches

Our search for ways to improve this situation has involved a comparative analysis of six alternative approaches. Although these are not the only gap-bridging approaches, these approaches that has been chosen based on literature review and experiences on what can be used for automating the process of providing sufficiently complete, consistent, traceable, and testable requirements specifications. These approaches, except computer-initiated template, are relevant to current industry practice and are available.

Template-based Refinement. The idea of using templates for expressing requirements is based on the use case templates by Rumbaugh [34] and Cockburn

[12]. Other similar works are the Volere Requirements Specification template [32] and the User Requirements Document Template developed at CERN [11].

Requirements templates help to express requirements. Requirements information is structured in a fixed form, so requirements engineers know what missing information must be searched and reuse is promoted.

Human-initiated Template. The requirements engineers detail the textual properties of each requirement. There are many usage of this type of template such as use case template, Volere software requirements template, and MBASE SSRD templates. Most of them are documented using either word processing or requirements management tools. On the other hand, human-initiated templates can be used in any domain [32].

Computer-initiated Template. Computer-initiated template approach seems a futuristic expectation as the requirements engineering activities heavily depends on stakeholders' interaction and communication. However, requirements specification can be analyzed and similar patterns for requirements related in a domain can be extracted and used in later projects. Although, fully automatic computer-initiated template is not applicable, partially filled templates will provide real help to the requirements engineers in producing requirements for a project.

Natural Language Processing (NLP). Detailed software requirements need to be written in a form that is understood by both the customer and the development team. Using the customer's language to describe these software requirements is most effective in gaining their understanding and agreement. That's why the natural language is widely used in industry to state requirements for all types of systems, because it is flexible and universal. A recent study shows that several software development companies use common natural language for specifying requirements in the early phases [27]. Although documents written in natural language are often hard to process automatically and inconsistency is difficult to detect, NLP techniques and tools [1; 10] can be used to extract template-relevant meaning from the stakeholder expectations to achieve a higher level of understanding.

Keyword Analysis. A keyword analyzer can be used to produce summary information to facilitate specification development. Specific words and phrases can be identified and searched in the requirements document for different purposes (e.g., checking the quality of document, defect detection, idea grouping, etc.) [37].

Formal Method Experts. Formal methods support precise and rigorous specification of those aspects of a computer system capable of being expressed in the language [22]. Formal methods can be particularly useful during the early stages of software engineering, and enable the software developer to discover and correct errors that otherwise might go undetected, increasing the quality of the software, while decreasing its failure rate. In addition, formal specifications are extremely useful in identifying inconsistencies in requirements specifications. However, it requires experts to refine the appropriate template elements into formal specifications before tools can be used to analyze them.

Inspections. Early detection of problems in requirements has a very important impact on the overall quality of the software development process. Formal technical reviews (inspections, walkthroughs, and technical reviews) are a method of removing defects at the requirements phase where they are introduced. The objective being, for the stakeholders, to detect errors or potential problems as early as possible in the system's description to ensure development proceeds in the right direction and meets the needs of the business [21]. Checklists for work products (e.g. requirements, and design specifications) are typically lists of issues that should be analyzed to ensure consistency, correctness, and completeness. The primary purpose of the checklists is to provide the reviewer with hints and recommendation for finding defects.

3.1. Evaluation of Gap-Bridging Approaches

We have analyzed each of these alternative approaches with respect to five criteria that we consider highly important with respect to our goal: Required human resources, Available technology, Generality, Scalability, and Precision. The results of the analysis are summarized in Table 1. The ratings for terms of the bridging approaches support in achieving the criteria were made on a scale of zero to four stars: 0–weak; 1–modest; 2–intermediate; 3–strong; 4–very strong. “Required human resources” shows how much resources needed to use the approach. “Available technology” shows if there are any supporting tools and existing technology used. “Generality” shows the quality of having general applicability in different application domains. “Scalability” shows if an approach is equally well suited for small and large projects. “Precision” shows how precise information the approach will provide.

The ratings were assigned based on the weights of the criteria on the strengths and limitations of the

approaches and their applicability in the software development. The two different stars for a cell show the range of those criteria since in different domains or with different techniques some partial solutions are feasible. As a conclusion, no approach dominates all of

the other approaches with respect to all of the criteria. This suggests that hybrid combinations of the approaches will be attractive. For example, inspections could be combined with computer-initiated templates and NLP.

Table 1. Evaluation of Gap-Bridging Approaches

	Required Human Resources	Available Technology	Generality	Scalability	Precision
Human-Initiated Template	** Template-building and refining process is slow, labor-intensive.	**** Can be used with formatted specifications. Strong tool support. [29; 32]	**** Can be used for any domain. [32]	** Feasible for any size, but with diseconomies of scale. [29; 32]	** Builds a good basis, but introduces some misinterpretations and defects.
Computer-Initiated Template	** *** Some effort required for post-processing entries.	* ** Currently infeasible to fully infer meaning from text. Some domain-model checking feasible for narrow domains.	* ** Some entries feasible (e.g., priorities) others only feasible for narrow domains.	* ** Weak for large complete solution, partial solution scalable.	* ** End results depend on human expertise in post-processing entries.
Natural Language Processing	** *** Depends on pre- and post-processing human resources. Best in narrow, mature domains. [1]	* ** Currently infeasible to fully infer meaning from text. Some domain-model checking feasible for narrow domains. [28; 33]	* Only feasible for mature, narrow domains with restricted vocabularies. [26]	* Weak for large specs, broad domains, large vocabularies.	* ** End results depend on human expertise in post-processing entries. [14]
Keyword Analysis	** Preprocessor could help focus on common terms, synonyms. [37]	**** Numerous keyword analysis packages available. [37]	**** Less valuable in domains with rapidly changing vocabularies.	* *** Processing scalability is strong. Output value decreases with size, breadth of domain.	* ** End results limited, depend on human expertise in post-processing entries. [38]
Formal Method Experts	* Formal method experts scarce. Would also have to consult domain experts. [19]	** Formal spec languages available, some with moderate tool support. [18]	*** Applicable to most domains. Harder to formalize levels of service.	** Formal method experts scarce. Larger specs harder to formalize. [20]	**** Experts produce most precision in specs. [19]
Inspections	** More people involved early, but with effort savings later. Some effort to post-process into template. [2]	**** Mature practices. Special techniques such as perspective-based reviewing effective [2; 15]	**** Can be used for any domain. [15]	** Feasible for any size, but with some diseconomies of scale.	*** Produces strong-shared vision, earlier defect elimination, reduced rework. [21]

Both the inspections and human-initiated template approaches are fairly strong across all the criteria. Inspections provide some comparative advantages in early defect reduction, reduced rework, use in avoiding misinterpretations, and use in early elaboration of the stakeholders' shared vision of the proposed system.

Some shortfalls of inspections (e.g., extra post-processing effort to convert the inspection results into formatted specs) could be reduced via hybrid approaches. For example, computer-initiated template could be generated and used as the artifacts to be inspected, ensuring that the inspection fixes would yield formatted specs directly. Keyword analysis could be applied as a generator of useful subsidiary artifacts such as lists of elements including common terms and word-association diagrams. Formal methods could compatibly be applied for specs needing high precision. Natural language processing has limited applicability in its current state, but can be valuable for domain – specific model checking in such mature and

focused domains as telecommunications or aircraft control.

Overall, the analysis indicates that hybrid methods have promise as a basis of improving over current human-initiated template approaches to bridging the pre-requirements to requirements gap.

4. Our hybrid method

The Requirements Engineering is a group problem solving process where many stakeholders make a variety of contributions that ultimately affect the effectiveness and efficiency of the software product. A major challenge is thus to understand this group process, and based on this understanding, find efficient ways of supporting groups of stakeholders in solving the problem of deciding what techniques to use. Requirements must be analyzed to determine completeness, consistency, and testability. In the

process of formalizing informal requirements, ambiguities, omissions, and contradictions will often be discovered.

Our hybrid approach (see Figure 1) for formalizing informal stakeholder inputs is automatically putting the categorized and prioritized EasyWinWin artifacts into a draft requirements template with previously defined criteria for template properties and using NLP, keyword analysis and inspections to analyze, control

and improve the quality of requirements specifications. By establishing a feedback mechanism using the data gathered from NLP, keyword analysis and inspections, the process can be improved incrementally by taking advantage of the defect information. Strong support for comments on the intermediate results allows stakeholders to continuously remove defects and improve the quality of the requirements.

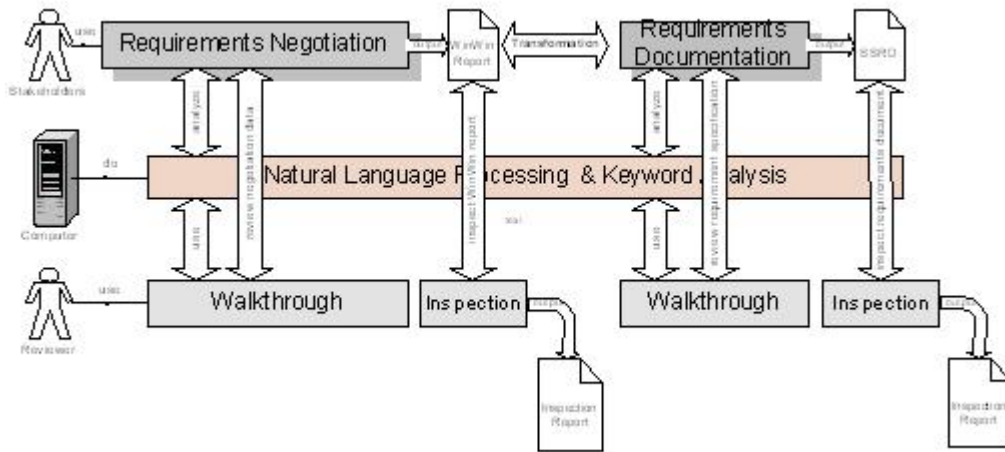


Figure 1. Hybrid method framework

Mixed-initiated template. Mixed-initiated template is the combination of human-initiated template and computer-initiated template. Requirements templates can be partially filled by computer processes by extracting information from the requirements negotiations results and rest of the work left to the stakeholders to fill and correct.

Different type of requirements templates can be generated for use. For example, in MBASE SSRD document, we have five different types of requirements: project requirements, capability requirements, system interface requirements, level of service requirements, and evolution requirements [7].

NLP. The approach we take to automating the analysis process is as follows: We collect a list of natural language checking criteria in terms of the parsing rules for a natural language parser. We then apply these rules to the parser output and feed the results back to the stakeholders for further correction.

Keyword Analysis. The words and phrases used as pattern indicators for respective categories were constructed. If a certain term or keyword occurred in the text of a specification, then as far as possible that specification belonged unambiguously to only one category.

Inspection. Because it costs so much more to fix defects later in the development process, formal inspection of requirements is perhaps the highest leverage software quality practice available. Combining formal inspection with incremental informal requirements reviews provides a powerful approach to building quality into the product.

There are three levels of checking during the process of requirements negotiation and specification that are listed below:

Table 2. Typical problems for a statement

Criteria	Problem	Solution
Complete	Missing information	Use NLP and inspection to check for missing information
	Incorrect statement	Use inspections for finding incorrect statement
Consistent	Internal consistency	Use same terms inside the definition of requirement
	No identifier	Assign unique identifier for individual statements
Testable	Unverifiable statement	Use inspection to find unverifiable statements
	Unclear terms /statement	Use inspection to find unclear terms and provide more measurable meanings
	Ambiguous term	Use keyword analysis and inspection to find ambiguous term

Statement-level checking: This checking is performed during converge on win conditions activity when a win condition identified. It is checked according to the problems we gather for a statement (see Table 2). In addition, the agreements are checked according to the same criteria during negotiate agreements step.

Table 3. Typical problems for a requirement

Criteria	Problem	Solution
Complete	TBDs	Based on attribute provide more meaningful comments
	Missing information	Use a template structure to check for missing information
	Nonexistent references	Use appropriate terms in each part of the requirement details
Consistent	Internal consistency	Use same terms inside the definition of requirement
Traceable	Missing reference	Add reference to its source
Testable	Not measurable; not testable	Provide measurable and easy to follow steps for testing

Requirement-level checking: This checking is performed while new requirements are transformed from agreements and new information added as details to the requirements template. Typical problems and solutions how to eliminate them are also provided in Table 3.

Table 4. Typical problems in a requirements document

Criteria	Problem	Solution
Complete	Missing requirement	Any external requirements imposed by a system specification should be acknowledge and treated
	Missing data	Specify the responses to both valid and invalid input values
	Missing label, reference or definition	Specify labels, references to figures, tables, and diagrams and definition of all terms
	Use of TBDs	Describe the conditions causing the TBD, what must be done to eliminate it, who is responsible for its elimination, and by when it must be eliminate
Consistent	Use different formats	Describe all requirements in same output format (i.e., tabular)
	Conflicted information	SRS should be reviewed by an independent party to identify conflicted actions so that it can be corrected
	Use different terms for an object	Eliminate multiple terms having the same meaning and use standard terminology and definitions
Traceable	Missing backward traceability	Each requirement explicitly referencing its source in earlier documents
	Missing forward traceability	Each requirement in the SRS have a unique name and reference number
Testable	Ambiguous requirements	Use concrete terms and measurable quantities

Document-level checking: This checking is performed both at the end of the negotiation process

and documentation process to improve the quality of the results and remove the defects still exist in the product. Typical problems and solutions are provided in Table 4.

The gap-bridging approaches for formalizing informal stakeholders' inputs assist stakeholders during the negotiation and documentation to achieve the complete, consistent, traceable and testable requirements specifications. Templates approach will be used to check completeness of the items of the requirements and assign traceability information to the requirements. Natural language processing techniques will be used to find incomplete statements and missing information in the requirements. Keyword analysis will be used to find and replace the ambiguous terms and related requirements that affect the consistencies of the requirements. Formal methods expert will be helpful to define the critical information in order to measure and test the requirements, especially for safety-critical systems. Inspections have many forms to be used for checking the criteria. Formal inspections will be used to check the criteria for the documents. In addition informal reviews with checklists help finding the problems in the statements and requirements.

Although we have proposing to use these techniques with requirements negotiation process, it is able to used with other requirements elicitation process by applying the NLP and keyword analysis on the initial data. However, it also has some limitations regarding the rules that can be used by NLP and keywords used by Keyword Analysis based on the application domains. On the other hand, after the initial effort reviewing and expanding these rules, they can be easily applied to similar projects. Another limitation is that the hybrid method is designed to assess gap-bridging approaches in use by the industry. In addition to using checklist-based reading which is chosen in order to partially automate the inspection that can assist the Independent Verification & Validation people, other inspection techniques such as perspective-based reading can be applied to the negotiation results and requirements documents.

5. Initial results

At USC, we developed the requirements for electronic services applications for the USC community as part of our software engineering team project course. During the course we conducted an experiment about how the teams formalize their requirements throughout the software life cycle. We conducted requirements negotiations involving real clients and 5-6 person teams of computer science

graduate students as developers. The goal of these projects is to negotiate functionality, budget, schedule, architecture, and transition for proposed e-services applications to be used at USC. We have been experimenting with EasyWinWin negotiation, the MBASE SSRD requirements template, a formatted specification template, and some of our gap-bridging approaches as a way of addressing the problem of formalizing our project requirements. We collected the requirements engineering related artifacts information (negotiation results, reviews on negotiation results, requirements documents, inspections on requirements documents, NLP and keyword analysis on negotiation results) for further analysis to automate the process. The threats to validity were uniformity of the projects and the subjects. Each project covered a different application with different client and technical characteristics. There were outlier performers who were either tremendously effective or in over their heads. In addition, NLP may not provide completely correct results on the inputs. So some non-defects can be also captured as defects, which require a human to make the final decision.

5.1. Software development process

Project teams used the MBASE software development process. MBASE involves early reconciliation of a project's success models (correctness, stakeholder win-win, etc.); product models (domain requirements, architecture, etc.); process models (waterfall, spiral, etc.); and property models (performance, reliability, etc.). It extends the spiral model in two ways:

1. Initiating each spiral cycle with a stakeholder win-win stage to determine a mutually satisfactory set of objectives, constraints, and alternatives for the system's next elaboration during the cycle.
2. Orienting the spiral cycles to synchronize with a set of cycle anchor points: Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability (IOC) [5].

For USC's e-services application projects, project teams had roughly 11 weeks in the fall semester to go from a short problem statement used in EasyWinWin negotiation to an LCO package and then LCA package. In addition, project teams used a domain model [5] included the system boundary, its major interfaces, and the key stakeholders with their roles and responsibilities. This domain model also established a domain taxonomy, which was used as a checklist and organizing structure for EasyWinWin

requirements negotiation process. Moreover, this taxonomy was used as the table of contents for the Requirements Definition, ensuring consistency and rapid transition from EasyWinWin negotiation and requirements specification.

5.2. Acquiring informal stakeholder requirements inputs with EasyWinWin

EasyWinWin is based on the WinWin requirements negotiation model and the GroupSystems, Inc. Group Support System. It 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 [6; 9]. Moreover, it helps elicit often-tacit values and knowledge, create a common vision among the stakeholders, and increase stakeholder involvement and interaction. The approach captures pre-requirements of success-critical stakeholders as win conditions, issues, options, agreements, and terms. Teams can use EasyWinWin throughout the development cycle to create a shared project vision, to develop high-levels requirements definition, to produce detailed requirements for features, functions, and properties, COTS acquisition and integration, COTS product enhancement, and to plan requirements for transitioning the system to the customer and user. The EasyWinWin process is described in detail in EasyWinWin Guidebook [16].

The negotiation process. The inputs to the projects EasyWinWin workshops were typically a mission statement outlining the high-level objectives of a project and another statement specifying the negotiation purpose. In each activity in this process the teams added details and increased precision. The teams started with reviewing and expanding the negotiation topics, which served as both organizing framework and a stakeholder checklist for validating completeness of negotiation results. Then stakeholders brainstormed on the project and contribute their interests. The resulting collection of stakeholder statements and ideas provided a starting point for elaborating win conditions and defining important terms of the project domain. Stakeholders also organized these win conditions into categories representing the major negotiation topics. After defining a glossary of project terms, a key prerequisite for all subsequent process steps, stakeholders rated each win condition according to two criteria: Business Importance shows the relevance of a win condition to project success; Ease of Implementation indicates perceived technical, economic, or political constraints of implementing a

win condition. Stakeholders examined the results of the prioritization and identified issues and options in several iterations. During the requirements negotiation, teams worked on initial prototypes based on their understandings that helped them identify further issues and option for many issues in order to come up with better agreements at the end. The teams continued to work until they achieved WinWin equilibrium (i.e., all win conditions and options are covered by agreements and there are no outstanding issues).

The outcome was a set of well-defined deliverables: (1) the topics of a negotiation organized in a domain taxonomy, (2) a glossary with definitions of key project terms, (3) a set of agreements providing the foundation for further refinements and plans, (4) open issues addressing constraints, conflicts, and known problems, as well as (5) decision rationale showing the negotiation history (comments, win conditions, issues, options, etc.). These deliverables provided the input for further refinement in the requirements specification and traceability in software development life cycle.

The domain taxonomy for the negotiation consists of the Table of Contents of the System and Software Requirements Definition (SSRD) used in MBASE method. It consists of project requirements (cost, schedule, development tools, support, etc.), system capability requirements (features and services),

interface requirements (to the user and to other systems), level of service requirements ('non-functional' aspects), and evolution requirements (most likely directions of system change and growth) [5; 26]. This reviewed and expanded domain taxonomy where each stakeholder win-win artifacts were categorized then used as the outline for the Requirements Definition document.

5.3. Analysis of defects in EasyWinWin negotiation results

During our software engineering class, we have independent verification and validation (IV&V) people whom are full time working professionals taking a distance learning course and reviewing artifacts produced by project teams. After the negotiation the IV&V students did a review on the negotiation results to find defects that make them incomplete, ambiguous, unverifiable and unclear. Some examples of defect types found in agreements and their indicators are shown in Table 5. Further analysis done after the reviews for investigating automatic detection of some of the defects and the patterns that can be used to find are provided in Table 5.

Table 5. Examples of defect types and automatic identification

Agreement	Defect Type	Indicator	Found by
<i>Searchable archive based on predefined criteria</i>	Unverifiable statement	Predefined	speech tag: adjective
<i>User authentication</i>	Unclear statement	not a sentence	speech tag: no subject and verb phrases
<i>Search by specific fields e.g. title, author</i>	Missing information	no subject	speech tag: start with verb
<i>More time is required to integrate with Z-bit or keep it as low priority</i>	Unclear term	Z-bit	speech tag: proper noun

Some of the defects related with WinWin artifacts, number of defects and comparison about the defect types found during the reviews are shown in Figure 2. The selected projects are Multimedia Archive projects in order to build patterns to capture defects. Moreover, we have found that more than one defect types can occur in the same agreement. The two most detected defect types are missing information and unverifiable statements where the statements don't contain the sufficient information in order to provide the meaning of the statements. These are the reasons for the unclear and ambiguous requirements. Our analysis showed that natural language processing and keyword analysis techniques could provide assistance to the stakeholders to eliminate defects as early as possible in the negotiation process.

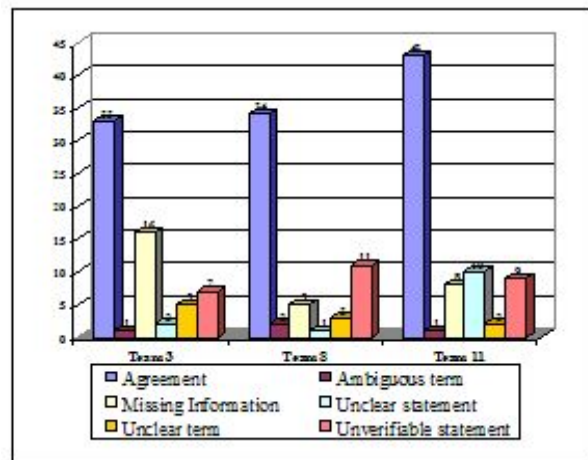


Figure 2. Number of defects in agreements

5.4. EasyWinWin output as software requirements definition

The teams had about three weeks to go from the EasyWinWin statements to a formatted specification. The agreements were used to identify the system requirements by mapping each agreement with one or more of the titles to the outline in the Requirements Definition. The MBASE provided templates for each type of requirements such as template format for capability requirements has the following attributes and more [ID number; Title; Description; Priority; Proposed Activity; Pre-condition; Post-condition; Reference; Risk Level]. An example of EasyWinWin agreement outputs mapped into MBASE SSRD requirements template is shown in Figure 3.

W9 LHF The server should have file management and user management function. (Taxonomy 2.1.2, 2.1.3.1)	→
A9 The server should have file management and user management function. (Taxonomy 2.1.2, 2.1.3.1)(W9)	→
W10 LHF The users have to be able to upload files, share files, within the group. (Taxonomy 2.1.2.2)	→
A10 The users have to be able to upload files, share files, within the group. (Taxonomy 2.1.2.2)(W10)	→

Requirement:	RC-2
Title:	Users can upload/modify/download/delete authorized files
Priority:	Very High
Description:	File management, as a part of the system, will improve information sharing and accessing. This will help users to upload/modify/download/delete files needed in their project.
Input(s):	User's request (click the files button)
Source(s):	User
Output(s):	Upload – the new files Modify – the modified files Download – the receive files in the user's own storage Delete – the file disappears in the project's storage
Destination(s):	Given storage of project group and user's own storage.
Precondition(s):	User know the files wanted to manipulate
Post condition(s):	User manipulates files
Proposed Activity:	File sharing and management as part of the collaboration services
Win Win Agreement(s):	A9, A10
Mainstream Scenario:	1. Upload files needed in their project; 2. Modify files needed in their project; 3. Download files needed in their project; 4. Delete files needed in their project.
Exception Handling Scenario:	If current user has no right to modify the file he is requiring, the system will give an warning information to show that he does not have enough right to do this operation.
Reference:	OCD 4.3 CAF-U2

Figure 3. EasyWinWin artifacts and MBASE SSRD requirements template

5.5 Inspection results of requirements document

Table 5 gives the results gathered from the inspections done by IV&V students on the requirements documents.

The defect numbers are the final defects registered by the author of the requirements documents. The reason why we provide two groups is that: Group A used Value-based checklist-based reading, whereas Group B used Value-neutral checklist-based reading [25]. The classifications of defect are missing, wrong, extra and open issue. Missing applies to things that should be in the requirement, but are not. Wrong is for things in the requirement, which are present but have mistakes. Extra is for things that are not called for, i.e.

Our analysis shows that the information captured during the requirements negotiations could be used to generate a draft version of requirements document automatically. When the traceability analysis was done between agreements and requirements, we saw that there are some agreements left without defining any requirement. On the other hand, there were some requirements without any trace to agreements, which means new requirements were added to the list from customer meetings where traceability information wasn't captured. This situation generally leads to an often-difficult decision problem: whether to maintain consistent EasyWinWin and SSRD versions of the requirements, or to treat the EasyWinWin results as a one-shot starting point for the SSRD.

Table 5. Number and type of defects found in requirements documents

		Team 3	Team 8	Team 11
IV&V in Group A	Defects	19	10	13
	Missing	6	0	7
	Wrong	13	7	5
	Extra	0	3	1
	Open Issue	0	0	0
IV&V in Group B	Defects	17	4	5
	Missing	12	3	0
	Wrong	4	1	5
	Extra	1	0	0
	Open Issue	0	0	0

there is no requirement that implies they should be present in the requirements. Open issue is a problem that the author cannot fix solely in the requirement or at that time.

Although, they used two different approaches to review the requirements documents, the defects they found gave us some idea about the total number and classifications of defects still exist in requirements specification. The number review results collected were 6: 3 in Group A and 3 in Group B. The total number of defects found is 68 and the average number of defects found is 11. However, there is no uniformity across all the review results. On the other hand, the main limitation of the review experiment was its coverage of reviews by individuals as compared to group reviewing.

6. Related Work

Because of software requirements specification being of central importance, many academic and practical work concentrates on which techniques to use to write good requirements such that they can be understood the same way by all stakeholders and transformed into a formal specification. Specriter [13] discusses a structured document generation and processing shell. Although it provides some limited consistency checking and specification in MIL-STD-490A format, it lacks ambiguity checking. The Requirements Apprentice [30] assists a requirements engineer to create a requirements specification from a set of informal requirements supplied by the user. However, it does not interact directly with the end-users.

The GMARC (A Generic Model Approach to Requirements Capture) [8] system transforms an informal set of requirements for an application into a requirements specification document. Although both of these tools generate a requirements specification document, they also required skilled requirements engineer to enter information.

The Requirements Counsellor [22] is a dual-use tool to analyze an informal requirements document and act as a mentor in writing requirements. It acts by identifying issues to be addressed by the writer and is based upon a collection of rules. However, it works on an initial requirements document instead of gathering stakeholder inputs and fixing the problems through specification activity.

The RECOCASE approach uses a CASE tool to assist requirements reconciliation. It starts with group brainstorming the functionality in the form of a use case diagram with representatives having viewpoints to

log and enter descriptions for use cases in a controlled language [31]. Although brainstorming and use case representation approaches are similar to our approach, the specification process requires users directly interaction with the system. On the other hand, we provide some automation to requirements template filling.

Kassel and Malloy [23] present an approach to partially automate the requirements elicitation and specification processes. Their approach allows all stakeholders interact directly with the system. Although the tool supports requirements engineers automatically create a draft requirements specification from stakeholders' inputs, it doesn't help analyze the requirements for completeness and consistency.

Most of the researches done on requirements specification start their processes with well-formed natural language requirements statements. However, the contribution of our research is that we start to use the techniques as early as possible in the software development process where stakeholders brainstorm, analyze, and negotiate on the problem to identify the requirements. Moreover, they continue to use these techniques further to transform their informal requirements into more structured and qualified requirements specification by using requirements templates.

7. Conclusions

In order to improve software engineering design, development, testing and quality control, first we have to improve the ability to articulate stakeholder-satisfactory requirements in an unambiguous testable format. However, this needs to be done in a way that avoids involving busy operational stakeholders in time-consuming wordsmithing and formalization sessions. The EasyWinWin tool provides an efficient and effective way to determine informal starting points for formal or formalized requirements.

This paper analyzed our experience on projects done during our software engineering course to evaluate different bridging approaches for formalizing such informal requirements inputs into more formal representations. After a comparison of relative strengths of target requirements representation methods, we discussed different gap-bridging approaches that we consider assist in getting more formal requirements. Then, we explained the hybrid method and the way we propose to use the gap-bridging approaches. After that, we described our software development process with the manual integration of the hybrid method, and summarized

some typical problems that exist and solutions to eliminate them by using those gap-bridging approaches.

Effective processes and automated tool support need to be developed through which requirements can be elicited, negotiated, documented, validated, and managed through the software development life cycle. Initial work shows that requirements captured in natural language can be evolved to a more formal representation by means of an intermediate activity in which the requirements are structured. The developers should conduct this activity and provide the results to the other success-critical stakeholders for review. Although it requires some additional effort during the negotiation, it is a useful step to improve the quality of EasyWinWin results, to discover open issues and problems in the requirements specification, and to establish useful links supporting traceability. Identified problems and links with the assistance of NLP and keyword analysis can be used in further refinements of the specification. By providing templates and guidance for transforming informal into more formal requirements, we are developing capabilities for faster and less intrusive elicitation of more complete, consistent, traceable, and testable requirements. Currently we are developing a tool that uses the information captures from the negotiation and analyze the results to eliminate defects, and later transform that information into structured requirements specifications. Further work is necessary to validate the hybrid method with the use of the tool.

8. References

- [1] V. Ambriola, V. Gervani, "Processing Natural Language Requirements", *Proc. of ASE 1997*, IEEE Press, 1997, pp. 36-45.
- [2] V. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Soerumgaard, and M. Zelkowitz "The Empirical Investigation of Perspective-Based Reading." *Empirical Software Engineering: An International J.*, vol. 1, no. 2, 1996, pp. 133-164.
- [3] B. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.
- [4] B. Boehm, P. Bose, E. Horowitz, and M.J. Lee, "Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach", *Proc. of the 17th ICSE*, 1995.
- [5] B. Boehm, D. Port, A. Egyed, and M. Abi-Antoun, "The MBASE Life Cycle Architecture Milestone Package: No Architecture Is An Island", *Proc. of the 1st Working Int. Conf. on Software Architecture*, 1999.
- [6] B. Boehm, P. Grünbacher, and B. Briggs, "Developing Groupware for Requirements Negotiation: Lessons Learned", *IEEE Software*, May/June 2001, pp. 46-55.
- [7] B. Boehm, D. Port, A.W. Brown, and E. Colbert, "Guidelines for Model-Based (System) Architecting and Software Engineering", USC-CSE, 2002.
- [8] D. Bolton, S. Jones, D. Till, D. Furber, and S. Green., "Using domain knowledge in requirements capture and formal specification construction", *Requirements Engineering: Social and Technical Issues*, Academic Press, 1994, pp. 141-162.
- [9] B. Briggs, P. Grünbacher, "EasyWinWin: Managing Complexity in Requirements Negotiation with GSS", *Proc. of the 35th HICSS*, IEEE Comp. Soc. Press, 2002.
- [10] J.F.M. Burg, *Linguistic Instruments in Requirements Engineering*, IOS Press, Amsterdam, 1997.
- [11] PSS-05 User Requirements Document Template, Technical report, CERN, 1998.
- [12] A. Cockburn, *Surviving Object-Oriented Projects: a Manager's Guide*, Addison Wesley, 1997.
- [13] S.C. Cook, "Knowledge Based Generation of Measuring Instrument Specification," *IMEKO Int. Symposium on Knowledge Based Measurement*, Karlsruhe, FDR, 1990.
- [14] F. Fabbrini, M. Fusani, S. Gnesi, G. Lami, "The Linguistic Approach to the Natural Language Requirements Quality: Benefits of the use of an Automated Tool," *26th Annual IEEE Computer Society – NASA Goddard Space Flight Center Software Engineering Workshop*, Greenbelt, MA, USA, November 27-29 2001.
- [15] M. Fagan, "Design and Code Inspections To Reduce Errors In Program Development." *IBM Systems J.*, vol. 15, no. 3, 1976, pp. 182-211.
- [16] P. Grünbacher, "EasyWinWin Moderator's Guidebook", JKU Linz, USC-CSE 2000, GroupSystems.com.
- [17] P. Grünbacher, A. Egyed, and N. Medvidovic, "Reconciling Software Requirements and Architectures: The CBSP Approach", *Proc. of the 5th IEEE Int. Symposium on Requirements Engineering*, Toronto, 2001, pp. 202-211.
- [18] J. Guttag, J. Horning, "Larch: Languages and Tools for Formal Specification", *Springer Verlag*, 1993.
- [19] A. Hall, "Seven Myths of Formal Methods", *IEEE Software*, 1990, pp.11-19.
- [20] A. Hall, "Using formal methods to develop an ATC information system." *IEEE Software*, 1996, pp.66-76.

- [21] M. Halling, St. Biffel, and P. Grünbacher, "An Economic Approach for Improving Requirements Negotiation Models with Inspection", *Requirements Engineering*, Springer, 2003.
- [22] L. James, "Providing Pragmatic Advice On How Good Your Requirements Are - The Precept "Requirements Counselor" Utility", *Proc. of the INCOSE 2000*, Minneapolis, Minnesota, 2000.
- [23] N. Kassel, B.A. Malloy. "An Approach to Automate Requirements Elicitation and Specification", *Proc. of the 7th Int. Conf. on Software Engineering and Applications*, November 3-5, 2003, Marina del Rey, CA, USA, pages 544-549.
- [24] H. Kitapci, B. Boehm, P. Grunbacher, M. Halling, and S. Biffel, "Formalizing Informal Stakeholder Requirements Inputs", *Proc. of the INCOSE 2003*, Washington DC, 2003.
- [25] K. Lee, B. Boehm, "Empirical Results from an Experiment on Value-Based Review Processes", *Proc. of the 4th ISESE*, 2005.
- [26] B. Macias, S. G. Pulman, "Natural Language Processing for Requirements Specification." (In) *Safety-critical Systems*, Chapman and Hall, 1993, pp.57-89.
- [27] L. Mich, M. Franch, and P.L. Novi Inverardi, "Market Research for Requirements Analysis using Linguistic Tools", *Requirements Engineering*, Springer, 2004, vol. 9, no. 1, pp. 40-56.
- [28] S. Nanduri, S. Rugaber, "Requirements Validation via Automated Natural Language Parsing." *28th Hawaii International Conference on System Science*, 1995.
- [29] U. Nikula, J. Sajaniemi, H. Kalviainen, "A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises." *TBRC Research Report 1*. Telecom Business Research Center Lappeenranta.
- [30] H. Reubenstein, R. Walters, "The Requirements Apprentice: Automated Assistance for Requirements Acquisition", *IEEE Transactions on Software Engineering*, 17(3), March 1991, 226-240.
- [31] D. Richards "Requirements Reconciliation using a Computer Supported Collaborative Approach", *Journal of Integrated Design and Process Science*, June 2003 7(2):113-129.
- [32] S. Robertson, J. Robertson, *Mastering the Requirements Process*, Addison-Wesley, 1999.
- [33] C. Rolland, C. Proix, "A natural language approach for Requirements Engineering." *Proceedings of the 4th International Conference on Advanced Information System Engineering*, 1992.
- [34] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999.
- [35] The Standish Group, "What are your Requirements?", West Yarmouth, MA, 2003.
- [36] K.E. Wiegers, "When Telepathy Won't Do: Requirements Engineering Key Practices" *Cutter IT Journal*, May 2000.
- [37] W.M. Wilson, L.H. Rosenberg, and L.E. Hyatt, "Automated analysis of requirements specification", *Proc. of the 19th ICSE*, Boston, MA, 1997.
- [38] W. Scott, and S.C. Cook, "A Context-free Requirements Grammar to Facilitate Automatic Assessment," in *Proceedings of 9th Australian Workshop on Requirements Engineering*, Adelaide, Australia, 6-7 December 2004.