Incorporating Innovation Into Iterative Software Development Using The Inventive Problem Solving Methodology

Ron Fulbright
University of South Carolina Upstate
800 University Way
Spartanburg, SC 29303
(01) 864-503-5683
rfulbright@uscupstate.edu

ABSTRACT

Iterative software methodologies allow development teams to be agile in their response to changing requirements and dynamic projects environments. Largely, however, the development team is limited to being reactive to requirement churn occurring outside the purview of the software development team itself. This paper describes a method based on the study of over two million patents, called inventive problem solving, allowing software development teams to be innovative, actively engineer changes to requirements, and discover new requirements by exploring alternatives to the problem solution.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *lifecycle, software development methodologies.*

Keywords

Software engineering, software development methodologies, iterative software development, spiral software development, agile software development, Rational Unified Process, Scrum, innovation, inventive problem solving, TRIZ, I-TRIZ.

1. INTRODUCTION

Over the last twenty-five years, iterative software development methodologies such as the spiral method, Rational Unified Process, agile software development, and Scrum have grown in response to weaknesses of the traditional sequential waterfall methodology. Iterative models incrementally evolve a software solution through repeated cycles of analysis, planning, prototyping, and review. The "design a little, build a little" characteristic of these methodologies mitigate risk and reduce waste because the solution continually changes following dynamic feedback.

All software development models contain some form of the basic lifecycle activities required to build and maintain a successful software solution: requirements extraction and definition, analysis, planning, design, coding, implementation, testing, deployment, verification, and maintenance.

Software solutions are constructed to solve a problem. Typically, before a software development project begins, the problem to be solved has already been identified and assumptions/decisions as to how to solve the problem have already been made. Consequently, the initial activities in a software development such as scope definition, business case, use case, risk assessment, etc. are biased toward the pre-defined overall problem solution architecture. As a result, software developers seldom "work on the problem" they instead "work on the solution."

At first, this may not seem problematic. What else is a software development team supposed to work on? However, what if there is a better way to solve the problem? What if the development team is building an inferior solution because a better approach goes unnoticed? It makes sense for the software development team to have a chance to explore alternative problem solutions early in the development process, even before, or at least as a part of, the project scoping and requirements definition phases. Since the software development team is better equipped to understand the technology, they are likely to envision alternative solutions no one else can. What is needed is a way to *innovate* about the problem solution during software development.

This paper proposes the use of a generic innovation technique called *inventive problem solving (IPS)* and suggests ways to incorporate the technique into iterative software development methodologies. The paper first describes iterative software methodologies and then presents details of IPS. An integration of IPS and the spiral methodology, the Rational Unified Process, and the Scrum methodology is proposed. A case study demonstrating how IPS yields a different set of software requirements for a collaborative crisis management solution is described.

2. ITERATIVE METHODOLOGIES

Although not the first iterative methodology conceived, Boehm's spiral model introduced in the mid-1980s marks a flex point in the history of software methodologies. Prior to the spiral model, the waterfall method was the predominant software methodology. Instead of a sequentially stepping through the development phases, the spiral model employed a "design a little, build a little" approach whereby the design of the final solution evolves along with the solution itself through multiple iterations of prototype and review [1]. As seen in Figure 1, development is depicted as a trajectory spiraling outward from the center in a clockwise direction through four quadrants, or phases. Each time around the cycle, the software solution gets a little more mature. Getting buy-in from users early in the development process and maintaining that support throughout development insures a desirable solution is being built. Note the first quadrant, the *determine objectives,* *alternatives, and constraints* phase, is where innovation should take place (alternatives).

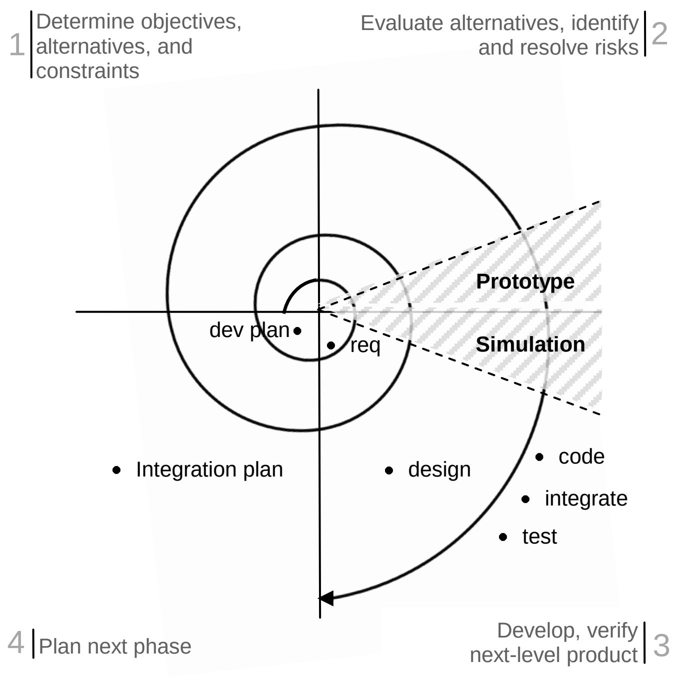
1 Determine objectives, alternatives, and constraints
Evaluate alternatives, identify and resolve risks 2
Prototype
Simulation
dev plan
req
Integration plan
design
code
integrate
test
4 Plan next phase
Develop, verify next-level product 3

Figure 1. The Spiral Development Model.

Throughout the 1990s, another iterative methodology evolved, the Rational Unified Process (RUP) [2]. The RUP is characterized by four phases: *inception, elaboration, construction,* and *transition.* Development iterates within each phase until certain completeness criteria are met. In each phase, different amounts of analysis, design, implementation, test, and deployment are performed. The RUP can be thought of as a superposition of waterfall and iterative methods. As expected, activities like business modeling, requirements definition, analysis and design are more prevalent in the earlier phases of inception and elaboration and this is where innovation should take place. The RUP is shown in Figure 2.

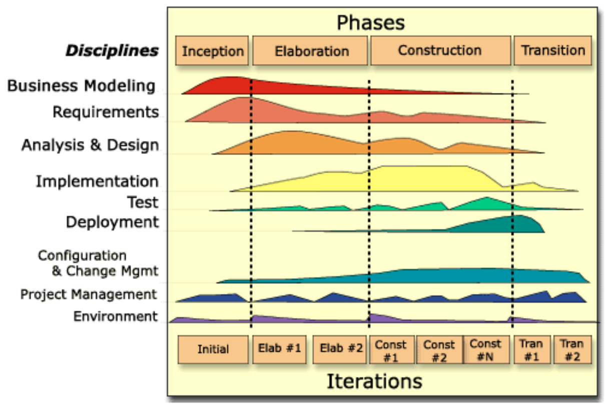

Figure 2. The Rational Unified Process.

In Scrum, another iterative software development methodology coming of age in the 1990s and in vogue in the 2000s, the notion of iterative development is embodied in timeboxed periods called *sprints*—typically 2-4 week time periods resulting in the full development of a portion of the software solution [3]. A smaller iteration, the *daily scrum* focuses team tasking during the sprint. At the beginning of each sprint, the *sprint planning meeting* chooses which requirements to focus development on during the sprint. The deliverable from each sprint is a piece of fully functioning software fulfilling the chosen requirements. Innovation should take place in the sprint planning meeting, the daily scrum, and in the original requirements definition. The Scrum methodology is depicted in Figure 3.

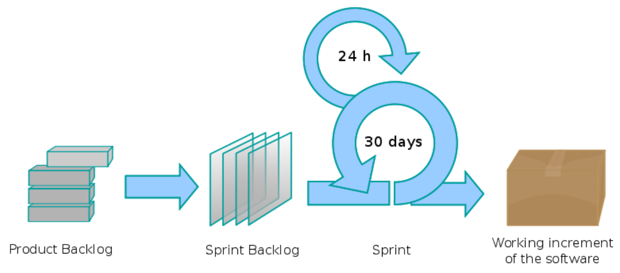

Figure 3. The Scrum Methodology.

All iterative methodologies accept the fact that the final solution cannot be known in its entirety before implementation begins because requirements constantly change. The goal of iterative software development methodologies is to make the development team as agile as possible and ready to respond quickly to such changes. However, rarely do methodologies give the development team the tools with which to *engineer the changes* and this is the promise of the inventive problem solving technique described in the next section.

3. INVENTIVE PROBLEM SOLVING

TRIZ (pronounced “trees”) is an acronym for the Russian phrase "Teoriya Resheniya Izobretatelskikh Zadatch" or “The Theory of Inventive Problem Solving” and dates back to 1946 when Russian engineer, scholar, and inventor Genrich Altshuller started reviewing patents looking for clues as to how inventive people solve problems [4]. Over the following four decades, TRIZ grew into nothing less than the science of technological evolution but was largely unknown to the Western world until the 1980s when some of Altshuller’s work was translated into English. This *classical era* of TRIZ saw the development of a number of tools and techniques designed to help practitioners inventively solve technical problems.

With the collapse of the Soviet Union in the late 1980s, TRIZ scholars and colleagues moved to other parts of the world where some have continued to extend the art and the science of TRIZ. One group, based in the United Stated, developed a modern extension called I-TRIZ (for “Ideation TRIZ”) comprising four methodologies [5]:

• IPS:	Inventive Problem Solving
• AFD:	Anticipatory Failure Determination
• IP:	Intellectual Property Protection
• DE:	Directed Evolution

AFD is a way of analyzing potential failure modes of systems and devising ways to prevent those types of failures. DE is a method of "inventing the future" and looking five and ten years ahead of the state of the art. IP contains ways to "invent the future competition" before your competitors do, and thereby protect yourself by already owning the intellectual property your competitors will have to invent to compete with you.

This paper involves IPS, a generic methodology enabling practitioners to innovate on demand about any type of system in any domain. At the heart of IPS is a database of over 400 *operators.* Each operator is an innovative concept gleaned from the study of over two million patents by TRIZ scholars. Practitioners use operators to stimulate thinking about ways to improve the system. Figure 4 shows the IPS methodology.

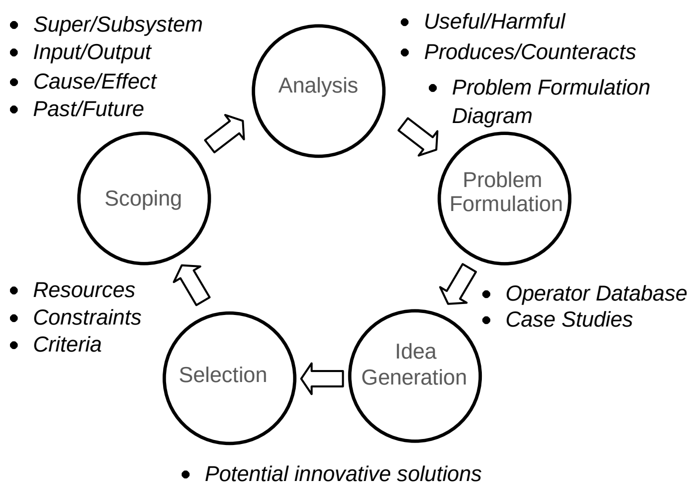

Figure 4. The IPS Methodology.

In the *scoping* phase, practitioners along with subject matter experts identify resources, constraints, and selection criteria. Combined with a systems analysis called the *8-way analysis,* the equivalent of a scope document (as used in traditional software development methodologies) is created. Through the *scoping* and *analysis* phases, practitioners create six different abstract descriptions of the system, each from a different perspective: *supersystem/subsystem, input/output, cause/effect, past/future, useful/harmful,* and *produces/counteracts relationships*. These descriptions are used to create a graphical representation of the problem domain known as the problem formulator (PF) diagram. The PF diagram captures the relationship between the desirable and undesirable characteristics of the system and exposes areas of the system most likely to benefit from an incremental change (an innovation). The PF diagram is used in conjunction with the *operator database* to generate dozens of potential innovative solutions.

An example of one of the operators is Add-a-marker:

Add a marker
Add a marker that can become the source of an easily detected field.

Adding radioactive dye to the bloodstream during an angiogram is an example of this concept in use (see Figure 5).

Figure 5. Adding a marker to the bloodstream.

The Add-a-marker operator is one of the concepts gleaned from the study of over two million patents. The theory behind IPS is that no matter what type of system is being studied one or more operators will be applicable and will likely stimulate a new idea.

Often, a combination of several operators forms an innovative solution. For example, consider the containment ring problem in a jet engine. The containment ring is a thick and heavy metallic shield preventing fragments from exiting the engine nacelle and damaging other parts of the aircraft in the event of a catastrophic failure of the turbine blades. However, the weight and bulkiness of the containment ring makes it difficult and expensive to remove and test as is periodically required. Dozens of potential solutions to this problem can be envisioned by applying various operators. For example, applying the operators:

- Segmentation
- Separation in time
- Separation on condition
- Introduce a liquid
- Add an intermediate layer
- Use a foam or empty space
- Abandon symmetry

yields a solution where non-uniform concentric ring arc segments containing a non-Newtonian fluid impact gel replaces the bulky containment ring as shown in Figure 6. For testing, the liquid is drained and the segments removed and tested individually.

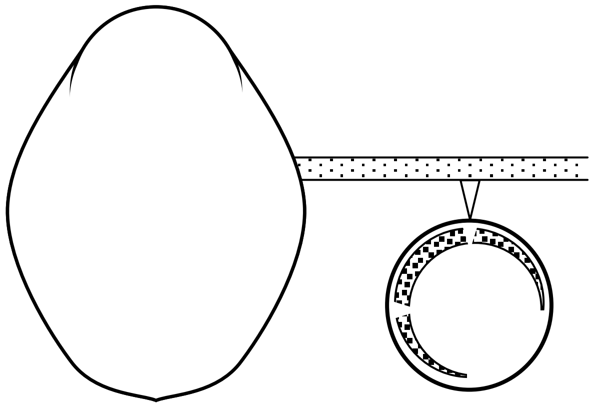

Figure 6. Innovative solution to the containment ring problem.

Having a summary of human innovative concepts available to us in the form of the operator database gives us the ability to apply the force of human innovational history to whatever problem we have at hand—including software.

4. THE CASE STUDY

Over a decade ago, the author was a senior software architect involved in a collaborative problem solving solution for a major oil company [6]. The development team utilized the spiral lifecycle model and created a successful solution that won the Microsoft Collaborative Application of the Year Award for 2001 winning over several hundred entrants worldwide. Recently, the author revisited the original requirements using the IPS methodology. Originally, this was done simply to demonstrate the use of IPS in the software engineering domain. However, systems analysis and problem formulation in the IPS methodology exposed key areas in the problem domain previously not considered by the original development team. If we had included these, our solution would have been superior.

When our client came to us originally, they described a "war room" problem-solving method which had been honed over several decades as an industry leader. When a crisis began, a conference room, or set of offices, were commandeered and turned into the central control hub dedicated to the management of the crisis and solution of the problem. All information and pertinent documents were collected and stored in this location and all communications were handled through this central command. Average problem resolutions took 3-5 days. Understanding the potential of the Internet, email, and World Wide Web-based communication, the company asked us to "create a virtual war room" to support global problem resolution and crisis management efforts. Their intent was to free themselves from the limitations of a physical war room and to reduce problem resolution time by at least 50%.

Therefore, when the development team began the design/prototype spirals, we sought only to flesh out the requirements of the "virtual war room." We never thought about making changes to the underlying crisis management process. Our solution simply "virtualized" the style of problem solving engrained into the corporation. When we acquired input from the user community, they dutifully described to us what they needed in the virtual space to do their jobs as they had been doing them in the physical war room.

Recently when IPS was used, it was apparent IPS "viewed" the problem domain from a perspective different than the user-centric perspective we encountered using the spiral model. IPS facilitated study of the underlying problem and resulted in the definition of several changes to the crisis management process itself. Once new approaches were identified, it was a relatively simple matter to envision additional features needed in the software solution. Not only did the innovative analysis show a better way to manage crises, it defined new requirements for the software solution. Figure 7 shows a portion of the problem formulator diagram resulting from the IPS analysis.

An example of a new feature discovered during innovative analysis was the "launch multiple simultaneous scenarios" concept. System analysis using the IPS methodology showed the dependency on sequential time-consuming processes. Crisis teams generally spent 1-3 days characterizing the problem and identifying experts in that problem domain to bring in on the resolution discussions. Only after a solution was agreed to was effort expended to get materiel and personnel moving toward the site of the crisis.

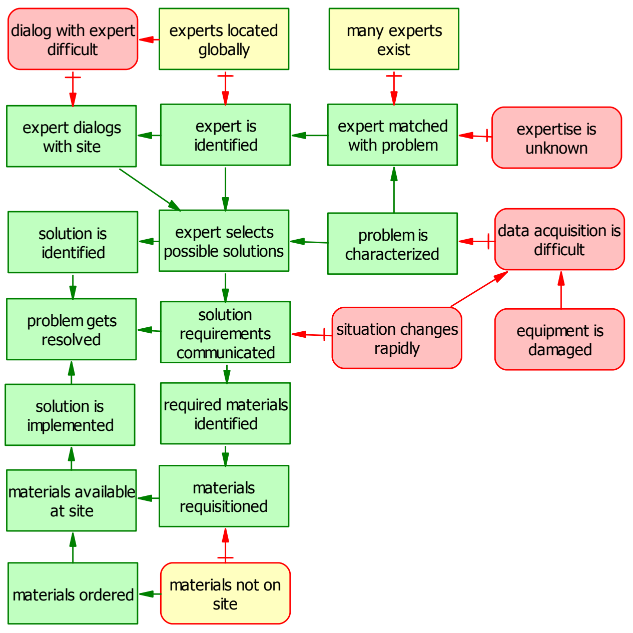

Figure 7. Problem Formulator Diagram (partial).

During analysis, IPS suggested consideration of the following operators:

- Duplicate critical elements
- Utilize time resources
- Partial action
- Preliminary action
- Apply multiple actions
- Multiply functions
- Preliminary dispensing
- Preliminary placement
- Modify time intervals
- Selective amplification
- Simultaneous operation

The combination of these concepts applied to the "sequential" nature of the company's crisis management process, suggested the idea of launching several different solution scenarios the moment a crisis begins. As more information about the nature of the crisis becomes available, the team can cancel those scenarios that do not apply. Even if a scenario must be altered for a particular crisis, getting it underway as early as possible could save as much as 2 days in problem resolution time. So rather than spending time collecting information and opinion to decide which resolution scenario to launch, the company should launch several, then use new information and expert opinion to "stand down" the scenarios that will not help. If the original software development team had done this analysis a decade ago, we would have built into the virtual war room several mechanisms directly supporting potential response scenarios, status tracking and reporting of response scenario activity, and "go/no go" polling on response scenario decisions.

Another example of a new feature suggested by the innovative analysis is a decidedly information technology-oriented idea. A major component of the company's crisis management effort was the characterization of the problem and the identification of the person, or persons, in the company possessing expert-level knowledge about critical elements of the problem. As the crisis developed and more information was obtained, users found great utility in the virtual war room's ability to connect easily and quickly with others. However, the virtual war room did little to assist in associating experts with the crisis itself. As a result, even with the virtual war room, expert identification time took as along as 2 days. During the innovative analysis, several of the same operators mentioned above suggested the idea that experts be "pre-matched" with various types of problems likely to be encountered. If the original software development team had conceived of this, we would have automatically indexed and categorized employees' emails, published papers and other documents, job roles, etc. to score their likely application to various kinds of problems the company had faced in the past. With such a tool built into the virtual war room, the crisis management team would have been presented with a list of "likely experts" every time a new piece of information was added. This would have reduced problem resolution time by 1-2 days.

The IPS analysis exposed different, but complimentary, requirements than the original development project encountered using the spiral model. When this was realized, the notion struck that IPS and iterative methodologies should be combined in some way.

5. INNOVATIVE ITERATIONS

IPS gives practitioners a way to identify alternatives to a system. For a software development team, the "system" is the software solution being developed and also the problem being solved by the software solution. Therefore, IPS gives development teams the ability to innovate about the software and the problem domain at the same time. For this reason it is reasonable to fit IPS into iterative methodologies during the planning, analysis, and definition phases.

Figure 8 shows an *innovate* wedge added to the first quadrant of the spiral model. This phase is where *objectives*, *constraints*, and *alternatives* are to be developed and these are certainly some of the outputs of IPS. Furthermore, other artifacts from IPS fit nicely into the first and second phase of the spiral model. Used early in the development process (the first or second spiral) will allow the development team to think of new ways to solve the problem, as was the case described above. Using IPS in later spirals will allow the development team to think of new ways to design and implement specific pieces of the solution.

Figure 9 shows the RUP methodology with an *innovation* discipline added. Increased amounts of innovation can be expected in the Inception and Elaboration phases with reduced amounts of innovation in the Construction phase. An initial increase in the Transition phase is expected as the team thinks of new deployment/maintenance strategies.

Figure 10 shows the Scrum methodology modified with an *innovation* process driving both the product backlog, the master list of requirements for the solution, and the sprint backlog, the list of requirement targets for the current sprint.

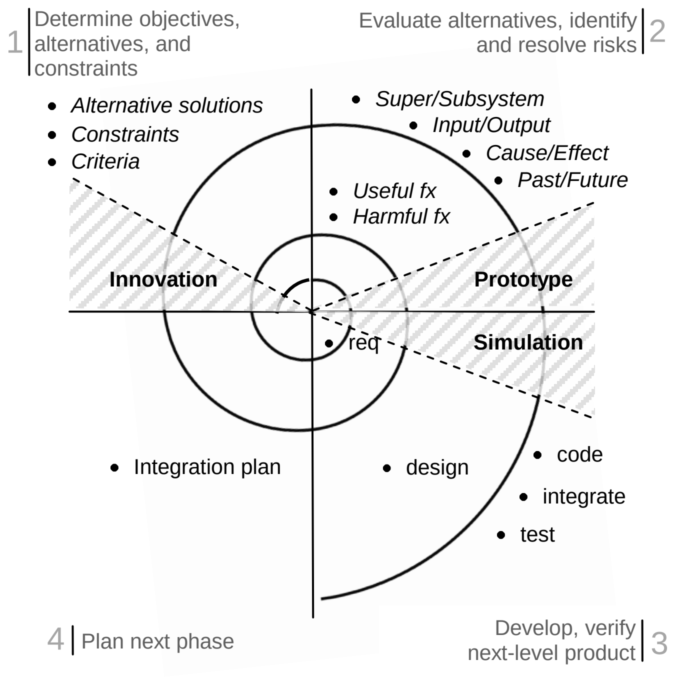

Figure 8. The Spiral Method With Innovation.

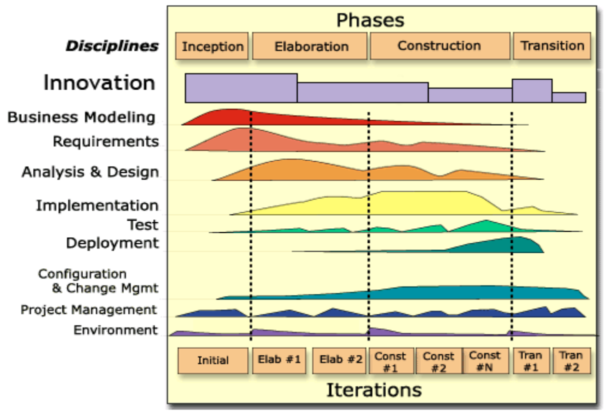

Figure 9. The RUP Methodology With Innovation.

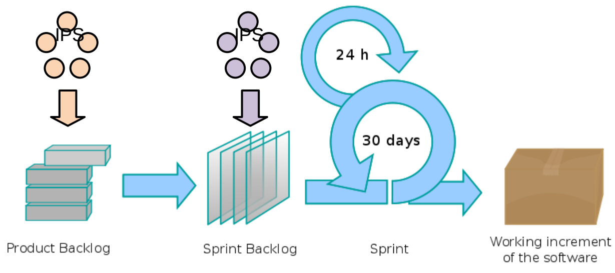

Figure 10. The Scrum Methodology With Innovation.

Two different IPS processes are shown because it is likely that different types of innovative analyses will be required for the product and sprint backlogs. The product backlog requires more problem domain innovation, such as that discussed in the case study, whereas the sprint backlog requires more innovation about the software implementation itself.

6. CONCLUSION

All of the iterative methodologies described in this paper, and many more not explicitly described here, enable a software development team to respond quickly to changing requirements. One goal of the iterations is to commit as few resources as possible to implementation before permitting more changes to the requirements. However, methodologies poorly address *how* changing requirements are identified nor give development teams tools with which to discover new requirements.

IPS provides a technique to explore alternative ways to solve the problem being addressed by the software solution and also a tool to explore alternative ways to implement the software solution. Most tools in software development methodologies just give developers ways to design the software and take as a starting point various assumptions and decisions already made before the software engineering process begins.

As shown in the case study, innovative analysis allows the development team to extend itself beyond the software system being developed and explore the original problem domain. In this way, incorporating innovative analysis into iterative software development methodologies gives development teams a way to *engineer* changes to the requirements in a way previously not possible.

7. REFERENCES

[1] Boehm, B, A. 1986. Spiral Method of Software Development and Enchancement. *ACM SIGSOFT Engineering Notes.* 11, 4 (August 1986), 14-24. Figure 1 is reproduced from this paper.

[2] Krutchen, P. 2003. *The Rational Unified Process-An Introduction.* 3rd Edition, Addison-Weseley. Figure 2 is a public domain image obtained from http://en.wikipedia.org/wiki/Image:RationalUnifiedProcess.png, April 2010.

[3] Schwaber, K., Beedle, M. 2002. *Agile Project Management with SCRUM.* Prentice Hall. Figure 3 is a public domain image obtained from http://en.wikipedia.org/wiki/File:Scrum_process.svg, April 2010.

[4] Altshuller, G. 1999. *Innovation Algorithm: TRIZ, systematic innovation and technical creativity.* Technical Innovation Center, Worcester, MA.

[5] Zusman, A. 1999. Roots, structure, and theoretical base. *TRIZ in Progress: Transactions of the Ideation Research Group.* Roza, V. ed. Ideation International Inc., Southfield, MI.

[6] Fulbright, R. 2001. *TeamSuite Feature Description.* ECMS/IT Factory Technical Report MSEC-061001-A-RDF. Available from the author.