

Scott R. Armstrong
 Pattern Project
 CS527 Fall 2006

REQUIREMENTS ITERATION



Context:

Requirements discovery and specification is an integral phase of every software engineering process. It is the problem description phase, usually results in requirements artifacts, and becomes the foundation for the rest of the phases in a software engineering and development process—problem analysis, architecture, design, implementation, etc. Requirements artifacts can take many forms: text documents, Use Cases, State Machines, Data Flow Diagrams, UML and other models. In Extreme Programming (XP) terms, the User Stories play the role of the initial artifact in the XP requirements phase. The requirements phase of a software development process identifies the problem that the software is intended to solve and specifies in as much detail as possible the desired functional and quality attributes of that solution.

Iterative processes have been acknowledged as more being effective than linear or single-pass processes. Iteration enables refinement, review, enhancement, improvement, simplification, etc.

Regular, planned, iterative review of the requirements throughout the software engineering process will facilitate earlier and more accurate alignment of the rest of the software engineering process with the intended purpose of the project. REQUIREMENTS ITERATION also helps to refine the accuracy of the software requirements based on discoveries, design decisions or implementation decisions later in the software development process.

Problem Summary:

The process of discovering and specifying requirements often results in artifacts that support the analysis and architecture phases of a project, but those artifacts will be superseded by artifacts from the later phases: an architecture document, design models, and implementation code, for example. Our tendency is to focus on the current activities, and as a result, the more current artifacts will hold our attention and interest. Certainly, working code is the perceived goal of the process. However, too much focus on the technical implementation and too little on the system requirements increases the risk of writing the software to solve the wrong problem.

Identifying requirements-related risks as early as possible can reduce the cost or time of development, but not identifying the real requirements will result in a software solution that doesn't solve the right problem. Incomplete or inaccurate requirements are a significant risk to any software engineering project and can unnecessarily extend the development time, increasing

the overall cost of the system. Early mitigation of risks in the process will result in a more efficient and less costly solution—late mitigation of risks in the process will be very costly.

Activities related to the requirements phase of a project have been historically treated as the starting phase of the overall project. More recent renditions of the Waterfall Method of software development include feedback between the phases. The Rational Unified Process includes several tools and models to describe the different aspects of the problem to aid in defining and documenting the requirements. XP puts the developers in direct contact with the users to write, discuss and implement User Stories—the XP equivalent of requirements documents—and includes iteration as a key principle. Iteration is a fundamental feature of modern and agile processes, but the emphasis on the requirements development process isn't commensurate with the value of the requirements phase.

The need to accept changes to requirements holds a prominent position in the Agile Manifesto—second in the list of principles only to customer satisfaction. “Welcome changing requirements, even late in development.” [<http://www.agilemanifesto.org/principles.html>] Early or late in the process, current and accurate—even changed—requirements are important for the value of the software being developed.

Though the Agile Manifesto identified the need to consider changes to requirements regardless of the phase (early, late, etc.) in the software system's lifecycle, REQUIREMENTS ITERATION isn't yet an automatic practice. Eric Evans details the need for domain analysis as a significant part of a software engineering process. Regular, iterative domain analysis effectively reviews much of the system requirements from the perspective of the user's domain.

Solution:

Include a review of the software requirements using a regular and methodical iteration plan in order to verify that the development team is solving the correct problem, to discover incomplete requirements, to correct erroneous or inaccurate requirements, and to provide clear, accurate and complete purpose to all phases of a software development project. Iteratively review the requirements as established in your REQUIREMENTS ITERATION plan to reduce the requirements-related risks as early as possible.

Develop a REQUIREMENTS ITERATION plan to take advantage of the logical separations of roles in your organization's software engineering process. It may make the most sense to review the requirements at the start of each traditional phase of the software engineering process—architecture, design, implementation, testing, etc.—for processes with more traditional development team roles. More contemporary and non-traditional processes such as XP may not have clearly defined phases and would require a different trigger—possibly time, such as weekly or biweekly, or XP iterations—for REQUIREMENTS ITERATION. Regardless of the software engineering process used, your organization should develop a plan to include REQUIREMENTS ITERATION at regular intervals or steps throughout the development process.

During a REQUIREMENTS ITERATION, assemble your organization's REQUIREMENTS ITERATION team and review the requirements to verify that they clearly explain the problem domain, fully

articulate the system requirements, and accurately reflect the intended purpose of the software system. The team should consist of active representatives of all significant groups involved in the software engineering lifecycle for the system being developed, including the user group. As with all teams, smaller teams are more effective and efficient than larger teams.

The following examples are descriptions of software system projects including very general requirements, a description of the solution to satisfy the requirements, and an example of the value that can be gleaned from REQUIREMENTS ITERATION. The format is as follows:

(). Software System

Requirements Description

Description of the solution

Benefit of REQUIREMENTS ITERATION

Comments (if any)

I. Software System: Calculator Program

Requirements description: Develop a system that will output the results of a simple expression, given values and math functions.

Description of the solution: a command-line program that inputs values and functions in a specific order and returns the results, a GUI that includes fields for values and functions and returns the results when the user interacts appropriately with the GUI, or a GUI that simulates a handheld calculator, including buttons and display window.

Benefit of REQUIREMENTS ITERATION: this system is very small and doesn't require a development team—it can be easily written by one person in a short period of time. Requirements in this case are relatively simple and not as likely to be misinterpreted. As such, an iterative review of the requirements will probably yield little value. There is little benefit in using REQUIREMENTS ITERATION for small system development efforts.

II. Software System: Process Control System for a Chemical Weapons Destruction Facility

Requirements description: Develop a system that will control the movement of a chemical weapon round through the destruction process, starting with the introduction of the round into the process facility and ending with the removal of the empty, burned and non-toxic parts of the round from the process facility.

Description of the solution: The software solution used to control this process will be specified in great detail in response to physical, environmental, and chemical

engineering specifications associated with the process—move the round into an environmentally sealed space, cut off the top of the round above the level of the GB (Sarin) liquid, extract and burn the liquid GB in an environmentally sealed furnace at a specified temperature for a specified period of time, burn the parts of the round at a specified temperature for a specified length of time to remove any traces of GB, remove the remaining parts from the process facility, and ship them to a recycling facility.

Benefit of REQUIREMENTS ITERATION: The requirements are very precise and should be referenced during each phase of the software development process to ensure strict adherence. The requirements should provide the functional, quality and system test conditions, making REQUIREMENTS ITERATION very valuable in terms of certifying system compliance even during the testing and maintenance phase.

III. Software System: Computer System Inventory Database System

Requirements description: Develop a system that allows users to add, remove, update and read computer system inventory information, including system serial number, hardware components, operating system version, application software version, associated peripheral devices, system location, and any support work done on the system, as well as to search the database for historical support work, location, general inventory statistics, and software versions.

Description of the solution: Develop a client-server system using an SQL-compliant Database with normalized tables to store the object data and a client application to add, remove, update and read the inventory of data objects. The system should be able to generate inventory reports that are either all-inclusive or applicable to hardware, software or support work. Knowledge of the types of systems used in the organization will be important when determining the data object classes and desired functionality.

Benefit of REQUIREMENTS ITERATION: for a loosely specified problem such as this system, planned REQUIREMENTS ITERATION would bring representatives from each part of the development process together with the user at planned intervals, allowing review and clarification of the user's intended solution, and helping the developers create software to solve the right problem.

Comments: as the development of this project progressed, the initial requirements were described and documented by a group and presented by a single person to the developer. The developer presented these requirements to a database team. The team initially developed multiple database representations of the data objects, later refining their representations to a single database solution. The client application was given minor consideration as the problem was determined to be primarily a database system. When the database solution was presented to the original group of users, there were several aspects of functionality that were assumed by the users that weren't considered by the database developers. The users' requirements were filtered by the users' group representative and by the development team leader. The specifications existed, but were

not reviewed to verify that the team was developing the right solution. The development of this system has stalled.

IV. Software System: Placement Testing Integration System

Requirements description: Integrate the results from a computer-based placement test system with the college's student information administration system to automate the online registration and student self-service functions of the student information administration system. The placement test system and the student system have already been developed. The placement test system generates a very large and comprehensive set of data values for each student, including test results and placement categories based on logic that can be configured in the test system. The student system is middleware that is highly configurable, and it can be configured to import and use a variety of data values. The user initially required that a system be developed to transfer placement information from the test system (eligible to take Math 098, Math 120 or Math 120) into the student system to allow students to register for the appropriate classes.

Description of the solution: Create a system integration application to read placement category information from the test system and integrate that information into the student system for use in determining which classes a student is allowed to take. This solution requires regular synchronization between the student system and the placement test system to update any changes in class prerequisites or minimum required placement test scores.

Benefit of REQUIREMENTS ITERATION: as the complexity of systems increases, so will the complexity of the requirements. In this case, the solution involves two existing systems and the system being developed. The number of users is larger than the previous examples and the users may not all work in the same office. Communication between developers and users will be very important to make sure that the integration system provides the correct functionality for the users. As complexity increases and communications hinder the solution, REQUIREMENTS ITERATION will enhance the communication process between users and developers, making it more likely that specified requirements are valid and that the correct solution to the problem is implemented.

Comments: The effort to integrate computer-based placement tests was initiated by the domain users. The domain users also specified the expected behavior of the system, including a high-level description of the expected interaction between systems. This integration system was nearly impossible to design because of the lack of agreement among the users regarding descriptions of the expected functionality, the increasingly complicated logic that needed to be configured in the placement test system, and concerns of how that logic should be interpreted in the student system. After several meetings, we stepped away from the project, assembled a team of people representing all concerned groups, and questioned the reasoning behind the initial interface system specifications. This review and subsequent meetings with the same group resulted in a clear understanding of the fundamental requirements of the computer-based system and

how it related to current practice using a paper-based test. We discovered that the domain users had made the assumption that placement test logic should be configured within the computer-based test system. By separating our concerns and keeping decision logic on course eligibility in only one system, we were able to determine that simple test scores were all that we needed from the placement test system. The implementation system used to transfer data values between systems became extremely simplified, the test system was used only to provide test scores, and the registration logic remained in the student system. The ultimate solution to this problem could have been determined much sooner, had the developers and users established and implemented a plan to regularly review and revised the original requirements.

V. Software System: Enterprise Resource Planning System

Requirements description: Develop a system that will allow groups of people to manage the financial and business activities within a company, employing the company's business model in the solution. For the sake of this example, I'll assume that we're developing a completely new system instead of adapting existing solutions that are already on the market.

Description of the solution: Develop an n-tier client-server solution that employs a client application or Web client-server system to access middleware (possibly distributed) that is used to access one or more databases (possibly distributed). The system should allow financial transactions typical for that company, views of historical transactions grouped and ordered in ways that make sense to that company, and possibly communications or work-flow systems to automate some of the more tedious tasks of "doing business" at that company.

Benefit of REQUIREMENTS ITERATION: As the architecture is developed to identify the various higher-level modules, planned, iterative reviews of the requirements will aid in relating the domain analysis to the requirements, identifying the significant modules needed in the system, assessing the value of the various features, and determining the scope of the system. As the development process—including REQUIREMENTS ITERATION—continues, the clarity, or lack of clarity, of the company's business model should become more apparent with each REQUIREMENTS ITERATION, providing the designers and developers insights into how best to implement the various lower-level modules of the system. For very complex systems that involve ongoing domain analysis, regular, planned REQUIREMENTS ITERATION will help develop clear, accurate and complete requirements as the domain model develops, reducing the costly development risks early in the software engineering process.

Consequences:

Iteratively reviewing requirements throughout the development process of a large, complex system could lead to feature creep, making it more difficult to deliver a working system; however, rapid feature creep in any system development effort should be an indication that the requirements were initially incomplete, the problem domain was not sufficiently bounded, or that

the problem domain isn't well understood by the user. In such cases, REQUIREMENTS ITERATION may take the form of feature shopping.

Smaller systems will likely not benefit from REQUIREMENTS ITERATION, as the requirements will generally be easily understood and uncomplicated.

It will be difficult to apply REQUIREMENTS ITERATION to software development activities that revolve around domain analysis to discover first what a system should do. Development activities that start with no specified behavioral or quality attributes have no concrete requirements to review. Requirements in this case are developed over time, and only after a reasonably functional system has been developed will REQUIREMENTS ITERATION be useful.

Related topics:

This process will rely on patterns that define the activities in a requirements review process and that define how to communicate requirements changes to the development team and users. It would also be useful to use the UBIQUITOUS LANGUAGE pattern (Eric Evans) as well as related process patterns described in James Coplien's books (*Process Patterns*, and *More Process Patterns*) as appropriate for your software engineering process.