

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 functionality and qualities 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, 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 requirements phase is often presented early in textbooks and quickly forgotten (unless the book happens to be about requirements). 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 replaced 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, so the more current artifacts will hold our attention and interest. Certainly, working code is the perceived goal of the process. 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. 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 were historically treated as the starting phase of the overall project. The later versions of the Waterfall Method of software development included 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 discuss and implement User Stories. Iteration is a fundamental feature of such agile processes, but the attention given to the requirements isn't commensurate with the value of the requirements phase.

The need to always change requirements holds a prominent place 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, recurring 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 in 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 frequently to reduce the requirements-related risks as early as possible.

Upon starting a significant phase or step in a software engineering process, revisit the requirements and verify that the requirements clearly explain the problem domain, fully articulate the system requirements, and accurately reflect the intended purpose of the software system.

The following three examples are very brief descriptions of a software system project, a simplified description of the requirements, an overview of possible solutions to all or parts of the requirements, and an example of the value that can be gleaned from REQUIREMENTS ITERATION.

#### **I. Software System: Calculator Program**

**Simplified Requirements:** Develop a system that will output the results of an expression, given values and math functions.

**Simplified Solution Possibilities, also simplified:** 1) Command-line program that inputs values and functions in a specific order and returns the results; 2) A GUI that includes fields for values and functions and returns the results when the user interacts

appropriately with the GUI; 3) A GUI that simulates a handheld calculator, including buttons and display window.

REQUIREMENTS ITERATION: The intended solution should probably be described in the initial requirements, and iterative review of the requirements will probably yield little value other than examples for unit and system tests, but this is a very simple software solution relative to the following software systems.

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

Simplified Requirements: 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.

Simplified Solution Possibilities: 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.

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 even into the testing and maintenance phase very valuable in terms of certifying system compliance.

## III. Software System: Enterprise Resource Planning System

Simplified Requirements: 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.

Simplified Solution Possibilities: 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.

REQUIREMENTS ITERATION: As the architecture is developed to identify the various higher-level modules, iterative review of the requirements will help analyze the domain model, identify the significant modules needed in the system, assess the value of the various features, and determine the scope of the system. As the development process 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.

**Consequences:**

Iteratively reviewing requirements throughout the software development process could lead to feature creep, making it more difficult to deliver a working system; however, rapid feature creep 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.

With larger systems, REQUIREMENTS ITERATION may take the form of feature shopping, and if the time required to develop a large system is significant, new technologies may be adopted as new system requirements.