



CHAPTER ONE

REQUIREMENTS MANAGEMENT IN A PROJECT MANAGEMENT CONTEXT

Alan M. Davis, Ann M. Hickey, and Ann S. Zweig

Project success is the result of proper planning *and* proper execution. Fundamental to proper planning is making sure that the work to be performed by the project is well understood and that the amount of work is compatible with available resources. Requirements management is all about learning and documenting the work to be performed by the project, and ensuring compatibility with resources. A well-executed on-time project that does not meet customer needs is of no use to anybody.

Requirements

Requirements define the desired behavior of a system¹ to be built by a development project. More formally, a *requirement* is an externally observable characteristic of a desired system. The two most important terms of this definition are *externally observable* and *desired*. Externally observable implies that a customer, user, or other stakeholder is able to determine if the eventual system meets the requirement by observing the system. Observation here could encompass using any of the five senses, as well as any kind of device or instrument. Next, a requirement must state something that is desired by some stakeholder of the system. Stakeholders include all classes of users, all classes of customers, development personnel, managers, marketing, product support personnel, and so on. It is not so easy to determine

¹A *system* is any group of interacting elements that together perform one or more functions. The elements could be electronic hardware, mechanical devices, software, people, and/or any physical materials.

if a candidate requirement is a valid requirement from this perspective. In fact, the only way to make the determination is to ask the stakeholders. The word *desired* was chosen purposefully and is meant to encompass both wants and needs (see *Wants vs. Needs* later in the chapter).

Requirements Management

This chapter is all about how project managers and analysts manage requirements. *Requirements management* is the discipline of

- learning what the candidate requirements are—the learning aspects of requirements management are generally called *elicitation*;
- selecting a subset of those candidate requirements that are compatible with the project's goals, budget, and schedule—the selecting aspects of requirements management are generally called *triage*;
- documenting the requirements in a fashion that optimizes communication and reduces risk—the documenting aspects of requirements management are generally called *requirements specification*; and
- managing the ongoing evolution of those requirements during the project's execution.

On large projects, the individuals who perform requirements management are generally called analysts, requirements analysts, requirements managers, requirements engineers, systems analysts, business analysts, problem analysts, or market analysts. In companies that mass-market the products of their development projects, these individuals are generally within the marketing organization of the company. In companies that build custom products for their customers, these individuals are generally within either the marketing or the development organizations of the company. In IT organizations where the products of development projects are used within the company, these individuals are within the IT organization itself and interface with the internal customers, or are within the internal customer organization and interface with the IT organization.

On smaller projects, the project manager often performs a majority of the requirements management activities because these strategic activities are so critical to project success.

Requirements Management and Project Management

Much of requirements management can be thought of as part of (or preceding) project planning, because one goal of requirements management is the decision concerning *what* system is to be built. However, because needs of customers are often in constant flux, requirements must be addressed throughout the project. At project inception, the project manager is often intimately involved in defining requirements. Because any subsequent change to requirements affects project scope, the project manager tends to stay involved in the requirements management process throughout development.

Project management of requirements activities is unique among most project responsibilities because of two factors: (1) the strong customer focus and (2) the “softness” of the discipline. In most aspects of project management, the constraints upon the task are pre-

defined, known, and finite. The project manager's job is to control the project in such a way that the short-term and long-term project goals are achieved. In the case of requirements, none of that is true. The stakeholders who are the source of the requirements may not be available when needed. Even worse, their needs are constantly in flux. The very act of asking the stakeholders for their needs induces the stakeholders to conceive of new requirements hitherto not thought of. Every time a requirement is stated, the stakeholders will think of many more. Every time a prototype is constructed and demonstrated to the stakeholders, they will think of dozens of additional requirements. The phenomenon is likened to a continuous application of Maslow's hierarchy of needs. Every time any need is satisfied, more needs appear. Thus, the actual performance of requirements management causes the project to expand in scope.

Most activities being planned, controlled, and monitored by project management tend to appeal to the left side of the brain. Everything is (or should be) well defined, concrete, measurable, and to a large degree controllable. Requirements management requires a large dose of both left-side and right-side brain function. For example, the skills required to perform requirements elicitation primarily reside in the right side of the brain. Such skills deal with communication, feeling, and listening. On the other hand, the skills needed to record and manage the changes to requirements (including the use of so-called requirements management tools) reside primarily in the left side of the brain. These skills deal with specification, attention to detail, and precision. For this reason, requirements management is more like project management than like the other tasks performed by the individuals reporting to the project manager. Requirements management, like project management, require a very diverse set of skills.

Types of Requirements

We defined a requirement as an externally observable characteristic of a desired system. Although this sounds fairly specific, in practice requirements come in a wide variety of flavors and serve a wide variety of purposes. The following sections describe some of this richness.

User/Customer vs. System (Problem vs. Solution)

Some authors demand that requirements describe a problem purely from the perspective of the customer and must omit any reference to any solution system. Other authors demand that requirements specifically describe the external behavior of the solution system itself (IEEE, 1993). We have found that most practitioners divorce themselves from either extreme and recognize that as the requirements process proceeds, requirements naturally evolve from descriptions of the problem to descriptions of the solution. When requirements are stated in terms of the problem without reference to a solution, they look like this:

We need to reduce billing errors by 50 percent.

When requirements are stated in terms of the external behavior of the solution, they look like this:

The system shall provide an “audit” command, which verifies the accuracy of bills.

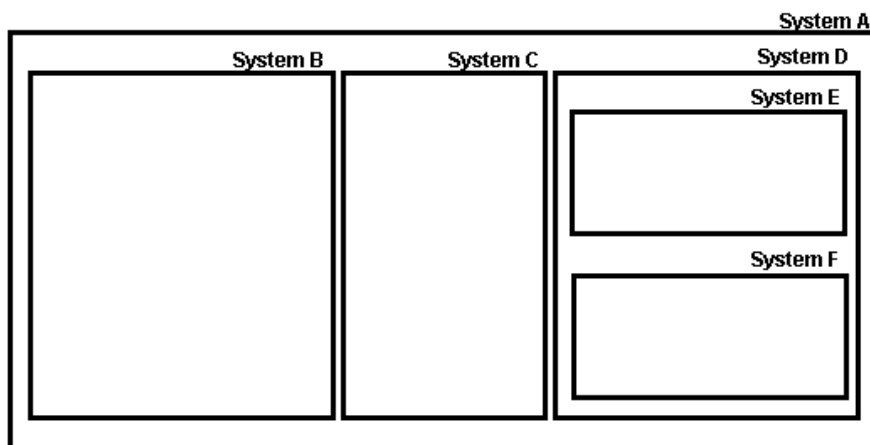
There is only a fine line separating the problem and the solution. In the preceding examples, one *could* argue that the former is actually within the solution domain. After all, reducing billing errors is just one way of trying to accomplish some real goal, such as increasing collections, increasing revenue, or maximizing cash flow.

Lauesen (2002) differentiates between user requirements and system or software requirements. He states that user requirements are supposed to address just the needs of the user, and system or software requirements are supposed to address the expected behavior of the solution system. However, he also correctly points out that in practice, most requirements describe external behavior of the solution system anyway, and that the term user requirements is generally applied to any requirements that are written in a language that users can understand.

Systems of Systems vs. Single Systems

By their very nature, systems are composed of other systems, as shown in Figure 1.1. For such systems, requirements are written for every system, usually starting with the top one. When requirements are written for the topmost system, they are written from a perspective outside that system, thus ensuring that all its requirements are externally observable. After these requirements are documented in a *system requirements specification*, system design (generally not considered part of requirements) is performed to decompose the system into its constituent subsystems and then to document those subsystems. Then requirements are written for

FIGURE 1.1. SYSTEMS ARE COMPOSED OF SYSTEMS.



each subsystem, from a perspective outside each of those subsystems, and the process repeats itself. As we get toward the lower-level systems, the system requirements are often replaced with two documents, a *software requirements specification* and a *hardware requirements specification*, each of which defines the requirements for its part of the system.

When a system is simple enough to not require decomposition into subsystems, the most common approach is to write a *system requirements specification* for the overall system, allocate each of the requirements to either software or hardware or both, and then proceed to write a software requirements specification and a hardware requirements specification.

When a system is composed entirely of either software or hardware, just one document is usually written—either a software requirements specification or a hardware requirements specification.

Primary vs. Derived

Thayer and Dorfman (1994) differentiate between requirements that are defined initially and requirements that are derived from those original requirements because of design decisions. For example, once the decision is made to include this requirement:

The system shall provide service x to the customers.

it becomes evident that we must also include this requirement:

The system shall bill the customers for using service x .

Project vs. Product

IEEE Standard 830 (1993) and Volere (Robertson and Robertson, 2000) make a clear distinction between requirements that constrain the solution system itself, for instance:

When the button is pressed, the system shall ignite the light.

and requirements that constrain the project responsible for creating the product, for instance:

The product must be available for commercial sale no later than April 2004.

IEEE Standard 830 calls the former *product requirements* and the latter *project requirements*. Volere differentiates between two types of *product* requirements: functional and nonfunctional; and three types of *project* requirements: project constraints, project drivers, and project issues.

Much agreement exists in the industry that product requirements are requirements, but little agreement exists concerning whether project requirements are really requirements. We happen to believe they are not requirements, but it is only a semantic issue. The fact is that during requirements activities, the team *will* need to perform trade-off analyses between both types of “requirements.”

Behavioral vs. Nonbehavioral

Some requirements describe the inputs into and the outputs from a system, and the relationships among the inputs and outputs. Others describe general characteristics of the system without defining inputs, outputs, and their interrelationships—that is, the functions that the system is intended to support. The former requirements are called *behavioral requirements*, although they have also been called *functional requirements* by the Robertsons (2000) and Davis (1993). The latter requirements are called *nonbehavioral requirements*, although they have also been called *developmental quality requirements* by Faulk (1997) and by the quite ambiguous and almost deceptive term *nonfunctional requirements*, by the Robertsons (2000) and Davis (1993).

Following are examples of behavioral requirements:

When the button is pressed, the system shall ignite the light. If the power is on and the on-off button is pressed, the system shall turn power off. When the user enters the command *xyz*, the system shall generate the report shown in Appendix H.

Examples of nonbehavioral requirements include all aspects of performance, reliability, adaptability, throughput, response time, safety, security, and usability, and they include such requirements as the following:

The system shall handle up to 25 simultaneous users. All reports shall be completely printed by the system within five minutes of the request by the user. The user interface shall conform to Microsoft standard *xxx*.

Wants vs. Needs

Many requirements writings seem to imply that one of the responsibilities of the analyst is to remove from consideration any requirements that are deemed to be “wants” rather than “needs” of the customers/users (IEEE, 1983; Swartout and Balzer, 1982; Siddiqi and Shekaran, 1996). Common wisdom and experience contra indicates this. Marketing studies have shown that people decide to buy or use a system because it satisfies their wants as well as their needs.

Requirements vs. Children of Those Requirements

When requirements are documented, they often are recorded more abstractly than is desirable, for example,

The system shall be easy for current system users to use.

This may be sufficient for early discussions, but it must be refined before the parties should agree to the effort. The most common way to do this is to document the refined requirements as subrequirements of the parent requirement, as in the following:

The system shall be easy for current system users to use.

- (a) The system shall include conventional keyboard and mouse.
- (b) The system shall exhibit the same “look and feel” of the existing legacy system.

Requirements should be refined whenever a discussion arises concerning the meaning or implications of a requirement.

Original Requirements vs. Modified Requirements

According to Standish Group Reports (1995), 58 percent of all requirements defined for software-based systems will change during the development process. According to Reinertsen (1997), a similar rate of change occurs for all products in general. This constant flux requires us to recognize that requirements evolve not only toward increasing detail but also toward altered functionality. We must clearly differentiate between requirements that were originally documented and requirements that become apparent only after development began.

Requirements in One Release vs. Requirements in Another

Almost all products evolve. Many requirements stated for, and implemented in, release n will undergo change in subsequent releases. This observation makes it clear that we must record the relationship between specific requirements and specific product releases.

Requirements Activities

Three distinct types of activities are performed under the auspices of requirements: elicitation, triage, and specification. The following subsections elaborate on these.

Elicitation

The first major set of activities within requirements management is called *elicitation*. Elicitation is the process of determining who the stakeholders are and what that they need—in other words, what their requirements are. Some of these needs can be “gathered”—that is, they are known and understood by the stakeholders, and all the analyst needs to do is “pick them up” from the stakeholder. Others may surface only as the result of stimulating the stakeholders; this type of activity most closely corresponds to the dictionary definition of “elicitation.” Other requirements need to be learned through study, experimentation, reading, or consultation with subject matter experts. Still others are discovered via observation. Regardless of the process used, and regardless of what the activity is called, the analysts must find out what the stakeholders needs are. Elicitation includes not just obtaining the needs but also analyzing and refining those needs to improve the team’s understanding of them. Once elicited, analyzed, and refined, these needs should be recorded as a list of candidate requirements, as shown in Figure 1.2

FIGURE 1.2. ELICITATION CREATES A LIST OF CANDIDATE REQUIREMENTS.

The user starts the RLM by placing it within the border of a defined lawn and pressing BEGIN MOWING from the Main Menu.

The RLM shall determine if it is in a defined lawn. If not, the RLM shall sound the error tones and display the message MOWER NOT IN RECOGNIZED LAWN on the first line and RETURN on the second line.

If correctly placed, the RLM shall beep once and wait for the user to step back beyond the safe distance range. After the user has moved beyond this range, the RLM shall move to a starting location within the lawn and begin mowing.

While mowing, the RLM's panel shall display nothing except in the event of an error condition, dump or refueling required, or an obstacle comes within the minimum safe distance.

The RLM shall check the grass height, grass type, grass density, and moisture of the lawn to determine the settings proper for cutting. Adjustments to the blade position and speed shall be made as required. When a swath is properly cut, the RLM shall move to an uncut area.

The cutting pattern shall begin with the perimeter of the lawn and work inward to the lawn's center. Each pass shall overlap the previous pass by a width less than or equal to 33% of the RLM's swath but greater than or equal to 25% of the RLM's swath.

This normal cutting pattern may be altered by obstacle avoidance maneuvers but shall resume when avoidance maneuvers are complete.

During avoidance maneuvers, the RLM may, for the sake of fuel efficiency, temporarily shut off its blades if over an area that has been properly cut. Obstacle avoidance is discussed in Requirement 510.

The RLM shall shut off the blades if fouling occurs to the degree that the RLM may damage itself. Should blade fouling occur, the RLM shall sound the error tones and display the message BLADES FOULED on the first line of the display. Should there be more than one blade . . .

The individual who conducts elicitation is generally called an *analyst*. An experienced analyst is adept at using a wide variety of elicitation techniques and possesses the sensitivities and skills necessary to assess the political, technical, and psychological characteristics of a situation to determine which elicitation technique to apply (Hickey and Davis, 2003; and Hickey and Davis, 2003a). Some of the classic techniques used during elicitation are as follows:

- *Interviewing* is the process of repeatedly prompting one or more stakeholders to verbalize their thoughts, opinions, concerns, and needs. The most effective prompts are open-ended questions, which force the stakeholder to think and respond in nontrivial ways. For example, prompts such as these are open-ended: "Would you please elaborate upon the problems you are experiencing now?" and "Why do you consider this a problem?" Other important aspects of effective interviewing include listening, taking notes, and playing back what you heard to verify that it was what was intended. Because over half of communication among individuals is nonverbal (Knapp and Hall, 1997), face-to-face interviewing is best. However, interviewing can also be performed over a telephone, though less efficiently. Gause and Weinberg (1989) provide a wealth of ideas on how to perform interviewing.
- *Brainstorming* is the process of gathering multiple stakeholders in a room, posing an issue or question, encouraging the stakeholders to express their ideas aloud, and having those ideas recorded somehow. The reason for demanding that ideas be expressed aloud is to encourage people to piggyback their own ideas on top of others' ideas. Criticism is generally discouraged. A wide range of variations of such meetings exists. Some variations

enforce anonymity via a tool; some have stakeholders record their own ideas, while others utilize a single scribe to record all ideas; and some discourage voicing the ideas aloud.

- *Conducting collaborative workshops* involves gathering multiple stakeholders together in structured, facilitated workshops to define the requirements for a system. Workshops may run from several hours to several days. During the workshops, facilitators lead stakeholders through a series of preplanned activities designed to produce the requirements deliverables needed. For example, participants may brainstorm on a variety of issues; create or review models, prototypes, or specifications; or negotiate and prioritize requirements. JAD (Wood and Silver, 1995) is probably the most widely known type of collaborative workshop, but there are many other variations, some of which use collaborative tools to increase efficiency (Dean et al., 1997). Gottesdeiner (2002) provides the best compendium of ideas on how to use collaborative group workshops for requirements elicitation.
- *Prototyping* is the process of creating a partial implementation of a system, demonstrating it to stakeholders, and perhaps allowing them to play with it. The bases for prototyping are that customers (a) can often think of new requirements only when they can visualize more basic requirements and (b) often can identify what they don't want more easily than what they do want. Davis (1995) provides the best overall summary of prototyping techniques and effects.
- *Questionnaires* are composed of series of questions that are then distributed to many stakeholders. Their responses are then collected, compiled, and analyzed to arrive at an understanding of general trends among the stakeholders' opinions. Unlike interviews and brainstorming, questionnaires assume that the relevant questions can be articulated in advance. For this reason, they are most effective at confirming well-formulated hypotheses concerning requirements, rather than assisting with the requirements synthesis process itself.
- *Observation* is an ethno-methodological technique where the analyst observes the users and customers performing their regular activities. In such cases, the analyst is passive and aims to not affect the activities in any way. It is the ideal technique for uncovering tacit knowledge possessed by the stakeholders. The best survey of techniques involving observation can be found in Goguen and Linde (1993).
- *Independent study* includes reading about problems and solutions, performing empirical studies, conducting archeological digs (Booch, 2002), or consulting with subject matter experts. Independent study is effective when others have addressed a similar problem before but the problem is relatively new to you.
- *Modeling* involves the creation of representations of the problem or its solutions in a notation that increases communication and provides fresh insights into the problem or solution. A wide range of modeling approaches exist, including object diagrams, data flow diagrams (DFD), the Unified Modeling Language (UML), Z, finite-state machines (FSMs), Petri nets, the System Description Language (SDL), statecharts, flowcharts, use cases, decision tables and trees, and so on. See (Davis, 1993; Kowal, 1992; Wieringa, 1996) for descriptions of most of these modeling notations. Although each provides the analyst with unique insights into the problem or its solution, the largest benefit often comes from using more than one. This is because each induces the analyst to ask (or answer) a certain class of questions, and the combination of multiple models induces more questions than the sum of using each one separately.

Triage

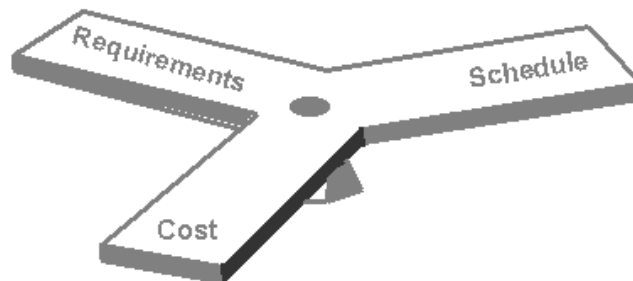
It is a rare project that has sufficient resources to address all the candidate requirements. To overcome this problem, project managers or teams need to conduct a scoping exercise typically called *triage*. Triage is the process of determining the appropriate subset of candidate requirements to attempt to satisfy, given a desired schedule and budget (Davis and Zweig, 1990; Davis, 2003). It is an activity conducted for an individual project that is quite similar to the performance of portfolio management, in which a set of projects are competing for the same finite set of resources and the project manager must choose from among them. See Chapter 2 in Meredith and Mantel (2003).

Triage is conducted in a formal meeting, usually led by the project manager, product manager, or independent facilitator. The participants must include representatives of at least three groups:

- *Primary stakeholders* need to determine the relative priority of candidate requirements and ensure that the voices of all classes of users and customers are expressed. Ideally, these representatives should be customers and users themselves, but often they are composed of marketing personnel, analysts, or subject matter experts.
- *Development* needs to be present to ensure that the requirements selected for inclusion in any release are reasonable relative to the realities of schedule and budget demands.
- *Financial support* must also be present. Otherwise, it is too easy for the other two parties to solve the triage problem by simply increasing available budgets.

Triage can be conducted by viewing the problem as one of balancing a multiarmed seesaw (see Figure 1.3). The three arms are the selected candidate requirements, the available budget, and the desired schedule. These three variables must be repeatedly manipulated until they are in balance. In this case, balance implies that there is a reasonably acceptable probability that the selected requirements can be satisfied by the project within the budget and schedule. Although the traditional development project manager's goal is to ensure

FIGURE 1.3. TRIAGE BALANCES A SEESAW.



completion on schedule and within budget, an even more responsible project manager takes a larger view. Just because the selected requirements *can* be built within the budget and schedule constraints does not mean that the project *should* be undertaken. A responsible project manager thus considers additional arms of the seesaw, which capture the risks associated with and the effect on achievement of business goals of the selected requirements. Thus, if the product is to be sold externally, additional arms include aspects of marketing, finance, personnel, and other factors as shown in Figure 1.4, adapted from Chapter 2 of Meredith and Mantel (2003). If the product is to be used internally, fewer factors must be considered, as shown in Figure 1.5.

The result of triage is a pruned version of the list shown in Figure 1.4. Although most practitioners think of this as a pruned list, a more reasonable way to visualize it is as the full original list, with each requirement annotated by whether or not it is included in the next release, as shown in Figure 1.6.

FIGURE 1.4. ADDITIONAL SEESAW ARMS.

-
- | | |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> • Marketing Factors <ul style="list-style-type: none"> – Size of Potential Market – Likely Market Share – Time Entering Market Window – Impact on Existing Products – Consumer Acceptance – Estimated Life of Product – Spin-Off Potential – Degree to Which We Understand Market • Financial Factors <ul style="list-style-type: none"> – Revenue Expectation – Profitability (Net Present Value) – Cash Flow Impact – Payout Period – Cash Requirements – Time to Breakeven • Personnel Factors <ul style="list-style-type: none"> – Training Needs – Labor Skill Needs – Level of Resistance | <ul style="list-style-type: none"> • Other Factors <ul style="list-style-type: none"> – Impact from Government Standards – Impact on Other IT Systems – Reaction from Stockholders (if a Corporation) – Reaction from Securities Markets (if Publicly Held Company) – Patent and Trade Secret Protection – Potential for New Patent Creation – Impact on Brand – Impact on Image with Customers and Competitors – Degree to Which We Understand New Technology – Ability to Direct and Control New Process – Experience We Gain from this Project to Be Applied to Future Projects – Average Order Size |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
-

Source: Adapted from Meredith and Mantel, 2003.

FIGURE 1.5. ADDITIONAL SEESAW ARMS FOR INTERNAL DEVELOPMENT.

<ul style="list-style-type: none"> • Demand Factors <ul style="list-style-type: none"> – Size of Potential Use – Customer Acceptance – Estimated Life of Product • Financial Factors <ul style="list-style-type: none"> – Increased Revenue Expectation – Decreased Cost Expectation – Cash Flow Impact – Payout Period – Cash Requirements • Personnel Factors <ul style="list-style-type: none"> – Training Needs – Labor Skill Needs – Level of Resistance 	<ul style="list-style-type: none"> • Other Factors <ul style="list-style-type: none"> – Impact on Other IT Systems – Degree to Which We Understand New Technology – Ability to Direct and Control New Process – Experience We Gain from this Project to Be Applied to Future Projects
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------










Specification

Once a subset of requirements is selected and agreed to by all parties, those requirements need to be refined and documented. This process is often called *requirements specification*.

Forms of Specification. A variety of common practices exist in the industry for documenting requirements, including the following:

- *A polished word-processed document.* Such a document typically follows one of the many standards available in the industry (e.g., IEEE, 1993 and Robertson and Robertson 2000) and is typically called a *software requirements specification* (SRS). Like all technical documents, it is composed of chapters and paragraphs. The biggest advantage of this approach is that all parties can read the document with a minimum of training. On the other hand, the biggest disadvantages are that (a) often many resources are expended polishing the noncritical parts of the document, (b) triage is almost impossible, (c) natural language can prove to be ambiguous, and (d) it is awkward to annotate each requirement *in situ*. This is a popular approach for constructing large embedded real-time critical applications, where “critical” usually means *life-critical*, *financial-critical*, or *security-critical*.
- *A hierarchical list of requirements.* Whether the list is packaged within the constraints of a formal SRS or not, it appears as a two-dimensional table, with each row corresponding to a single requirement and each column corresponding to an attribute of that requirements, including a unique identifier, the text, the priority, estimated development cost, and so on. The biggest advantages of this approach are that (a) all parties can read the list with a minimum of training, (b) fewer words means less time spent polishing, (c) triage

FIGURE 1.6. TRIAGE CREATES A LIST OF SELECTED CANDIDATE REQUIREMENTS.

The user starts the RLM by placing it within the border of a defined lawn and pressing BEGIN MOWING from the Main Menu.		YES
The RLM shall determine if it is in a defined lawn. If not, the RLM shall sound the error tones and display the message MOWER NOT IN RECOGNIZED LAWN on the first line and RETURN on the second line.		YES
If correctly placed, the RLM shall beep once and wait for the user to step back beyond the safe distance range. After the user has moved beyond this range, the RLM shall move to a starting location within the lawn and begin mowing.		YES
While mowing, the RLM's panel shall display nothing except in the event of an error condition, dump or refueling required, or an obstacle comes within the minimum safe distance.		YES
The RLM shall check the grass height, grass type, grass density, and moisture of the lawn to determine the settings proper for cutting. Adjustments to the blade position and speed shall be made as required. When a swath is properly cut, the RLM shall move to an uncut area.		YES
The cutting pattern shall begin with the perimeter of the lawn and work inward to the lawn's center. Each pass shall overlap the previous pass by a width less than or equal to 33% of the RLM's swath but greater than or equal to 25% of the RLM's swath.		YES
This normal cutting pattern may be altered by obstacle avoidance maneuvers but shall resume when avoidance maneuvers are complete.		YES
During avoidance maneuvers, the RLM may, for the sake of fuel efficiency, temporarily shut off its blades if over an area that has been properly cut. Obstacle avoidance is discussed in Requirement 510.		YES
The RLM shall shut off the blades if fouling occurs to the degree that the RLM may damage itself. Should blade fouling occur, the RLM shall sound the error tones and display the message BLADES FOULED on the first line of the display. Should there be more than one blade ...		YES

can be performed easily, and (d) it is trivial to annotate the requirements. On the other hand, the biggest disadvantage is that natural language can prove to be ambiguous.

- *Few or no documented requirements.* In this scenario, documentation of requirements is seen as a detractor from getting the product out. In effect, the code is the requirements, or more correctly, the code implies the requirements. The biggest advantage of this approach is that (in theory) no time is required to write or review the requirements, and thus total development time can be reduced by, say, 15 percent. However, this advantage does not come without the considerable risk of building the wrong product altogether. The proponents of this approach possess a variety of motivations. For example, some of those in the entrepreneurial world feel that getting to market fast with an innovative product is so critical to its market success they cannot afford to spend the time “investigating” the requirements—and they may be right! Meanwhile, those in the agile development community (Cockburn, 2002; Highsmith and Cockburn, 2001) claim that they build such small increments of the product, and if they make a mistake in such an iteration, it is easy to back it out and try again. Justification for recording requirements can be found in Hoffman and Lehner (2001).
- *The model is the requirements.* In some industries, requirements are not documented in natural language but are instead captured adequately in a model (see previous discussion of models). For example, in some business applications, a majority of the requirements can be captured using use cases, data flow diagrams, and entity relation diagrams. In some user-interface-intensive applications, a majority of requirements can be captured using use cases. And in some real-time systems, a majority of the requirements can be captured using Petri nets, finite-state machines, or statecharts. The unified modeling language (UML; Booch, 1999) is an attempt to capture all these models in one notation. The biggest advantage to this approach is that systems people (on the IT side and the customer side) can read the notations easily. The biggest disadvantages are that (a) nonsystems people on the customer side have difficulty understanding the notations; (b) no model is sufficient to represent *all* requirements, so they must be augmented in some way (for example, few of the aforementioned notations provide the ability to capture nonbehavioral requirements as described previously); (c) triage is likely to be difficult; and (d) it is almost impossible to annotate individual requirements.
- *The prototype is the requirements.* In this case, a prototype system is constructed and the customer likes it. Then the real system is constructed to mimic the behavior of the prototype. The biggest advantage to this approach is that customers can witness the intended system’s behavior first hand. The biggest disadvantages are that, (a) by definition, a prototype does *not* exhibit all the behaviors of the real system, so it must be augmented in some way, (b) triage is likely to be difficult, and (c) it is almost impossible to annotate individual requirements.

All of the approaches can be followed in an incremental manner (i.e., document a little, build a little, validate a little, then repeat) or a full-scale manner (i.e., document a lot, build a lot, validate a lot). Table 1.1 summarizes the advantages and disadvantages of the five approaches. In this table, notice that just because a technique has more check marks in its

TABLE 1.1. DISADVANTAGES OF VARIOUS REQUIREMENTS DOCUMENTATION APPROACHES.

Disadvantages	<i>Documentation Approach</i>				
	Document	List	Few/None	Model	Prototype
Natural language is inherently ambiguous	✓	✓			
Challenging for multinational efforts	✓	✓			
Notation not already known by customer				✓	
Difficult to annotate individual requirements	✓		✓	✓	✓
Difficult to select subset of requirements for inclusion	✓		✓	✓	✓
Insufficient to represent <i>all</i> requirements				✓	✓
Could imply unintentional requirements					✓
High risk of building the wrong product			✓		
Risk of incurring unnecessary up-front (perhaps nonrecoverable) costs	✓	✓		✓	✓
Could be challenging to maintain	✓	✓		✓	✓
Difficult to trace to origins and be traced from downstream entities	✓			✓	✓
Difficult to diagnose reasons for misunderstandings			✓		

column does not necessarily make it a worse approach; each comes with its own inherent risks. Only the project manager can decide which risks are worth taking.

As requirements are documented using any of the preceding approaches, disagreements will naturally arise concerning what individual requirements mean. In such cases, three solutions exist: (a) document the requirement in less ambiguous terms but using the same general approach, (b) supplement the requirement with another approach that has less ambiguity, and (c) refine the requirement into its constituent subrequirements, as described previously.

Attributes of a Specification. As work proceeds on requirements, they should evolve toward increased value to the project team. That means they should become less ambiguous, more correct, more consistent, and more achievable. For a more complete list of attributes that

requirements should exhibit see Davis (1995). The activities involved in determining if the requirements are evolving toward increased quality are generally called *validation and verification*, or V&V for short (Wallace, 1994). There appears to be some confusion within the industry concerning the differences between the two terms as applied to requirements, for example, see Christensen and Thayer (2001), Leffingwell and Widrig (2000), Wiegers (1999); and Young (2001). The confusion arises from the use of the terms in latter phases of system development. In later phases, *verification* of that phase's output is the process of ensuring that the output is correct relative to the outputs of the previous phase, and *validation* of that phase's output is the process of ensuring that the output is correct relative to the requirements (IEEE, 1986). Since requirements are usually considered the first phase of a system development life cycle, those definitions do not apply. However, if you consider that these words imply that verification ensures that the product is being built right and validation ensures that the right product is being built (Boehm, 1982), then we can extrapolate their meanings to requirements, as follows:

- *Requirements verification* ensures that the requirements themselves are written in a quality manner.
- *Requirements validation* ensures that the requirements as documented reflect the actual needs of the users/customers.

Then, to *verify* the quality of requirements, the following attributes must be addressed:

- *Ambiguity* is the condition in which multiple interpretations are possible given the identical requirement. Ambiguity is inherent to some degree in every natural-language statement. Thus, the parties can easily spend their entire project budget attempting to remove every bit of ambiguity. A more successful project will reword or refine a requirement only when the potential for adverse consequences is evident if the requirement stays as is. Another way to decide on whether a requirement statement is “good enough” is to determine if *reasonable, knowledgeable, and prudent* individuals would make different interpretations of the requirement.
- An SRS is *inconsistent* if it contains a subset of requirements that are mutually incompatible. For example, if two requirements are incompatible, or are in conflict with each other, then the SRS is inconsistent. Furthermore, an SRS should also be consistent with all other documents that have been previously agreed to by the parties.
- Requirements should also be *achievable*, which means it is possible to build a system with available technology, and within existing political, cultural, and financial constraints.

To *validate* requirements, the following attribute must be addressed:

- A requirement is *correct* if it helps to satisfy some stakeholder's need. Obviously, if a candidate requirement fails this test, it should be triaged out of the product.

Variations of Requirements Management Practices

Requirements management practices vary based on many aspects of the project. Let's look at some of these aspects and see how they effect requirements management.

Size of iterations

All product development efforts are iterative because as soon as customers start using any product, new requirements appear, thus driving another iteration. The differences lie in how big each iteration is and whether or not the team tries to satisfy “all the known requirements” in each iteration. As iterations increase in size (either in terms of elapsed time or sheer number of requirements), risks increase. In particular, the risks that increase include the likelihood of exceeding the budget, of completing after the desired delivery date, and of failing to meet customer needs. On the other hand, as iterations decrease in size, the effort for overhead tasks become a larger proportion of the total effort. With larger iterations, more effort must be expended during the requirements phases of each iteration.

Relationship of Iterations to Planning

In some cases, an entire product's requirements are explored and documented at project inception, and a product rollout strategy is developed that incorporates successively larger subsets of requirements in each iteration. In other cases, limited requirements activity occurs up front. The initial product is released primarily to acquire requirements feedback. Each successive iteration's requirements are defined based on the feedback acquired from the previous iteration.

Use of Throwaway Prototypes

Any iteration can be prefaced with the construction of a prototype. The purpose of the prototype is to remove the risk of building the wrong iteration. By seeing a prototype, stakeholders can provide valuable feedback concerning whether or not the development team is on the right track. Such an approach reduces the risk of the next iteration. When a prototype is used, minimal requirements effort is expended at project inception. Most requirements are defined after the initial prototype but before the development for the first real iteration begins.

Manufacturing Needed

Some systems require a manufacturing phase after development. This is primarily a function of the media involved. Pure software systems require no manufacturing (other than the trivial creation of CD-ROMs), whereas systems that include physical components do. When manufacturing is required, care must be taken during requirements elicitation and specification to ensure manufacturability and testability.

Research Needed

Some systems require research, invention, or innovation prior to starting the development activities. Usually, requirements are difficult to express when innovative research is needed. In such cases, a set of goals is stated (which are rarely termed requirements). Then the research is performed. Requirements efforts do not commence in earnest until after the research effort is complete.

Management Demand for Sequentiality.

If management enforces the idea that no task may be started until the previous task is completed, then elicitation must be completed before triage begins, and triage must be completed before specification can begin. Only the most conservative of management organizations still adhere to this ancient custom.

Iterative Nature of Requirements Process Itself

Hickey and Davis (2002) describe requirements as an iterative process where each iteration uncovers additional requirements, and changes the current situation. These changes to the situation, and the new requirements uncovered, drive the analysts to modify their approach for the next iteration. This is a more realistic view of the requirements process than attempting to do all elicitation on one phase.

Software-Intensive Applications

Traditionally, software had been developed using large iterations, with all the planning up front, with the assumption of high sequentiality. This approach was termed the *waterfall* model. It is typically represented by a linear PERT chart, as shown in Figure 1.7. Figure 1.8 shows where the requirements activities are performed during the development.

More modern software development projects use the so-called iterative model of software development (also called incremental). There are two general ways to plan the requirements for each iteration: by fixed time and by logical functionality sets. In the former, the length of time for each iteration is set in advance, and then the requirements are managed to ensure that only those requirements that can be satisfied in that time frame are included. Iteration length varies typically from a few weeks to a few months. In the latter way, logical subsets of requirements are grouped together and each iteration is scheduled to be reasonable with respect to the functions it is satisfying. In either case, the iterative method is typically represented as shown in Figure 1.9. Figure 1.10 shows where the requirements activities are performed during the development.

A more recent approach to software development is generally called *agile*. The agile movement (Cockburn, 2002; Highsmith and Cockburn, 2001) proposes a significant decrease in the power of project management and general management, and instead pushes many responsibilities down to the individual contributors. Readers wishing to learn the details of agile development should refer to the sources cited in the previous sentence. Here we discuss the implications of agile methods on requirements management itself. Instead of attempting to elicit requirements at the beginning of the development process, agile devel-

FIGURE 1.7. A WATERFALL MODEL.

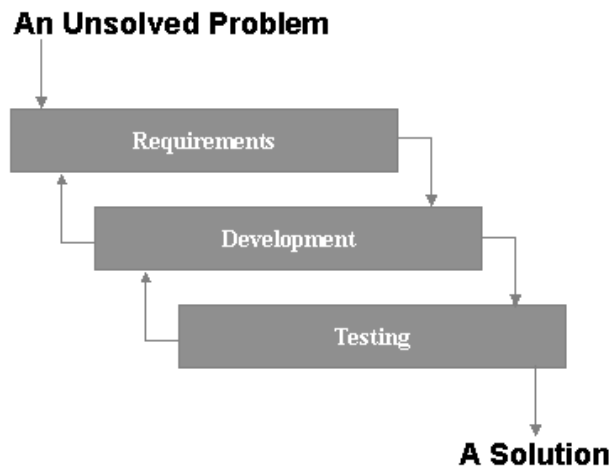


FIGURE 1.8. REQUIREMENTS ACTIVITIES WITHIN A WATERFALL MODEL.

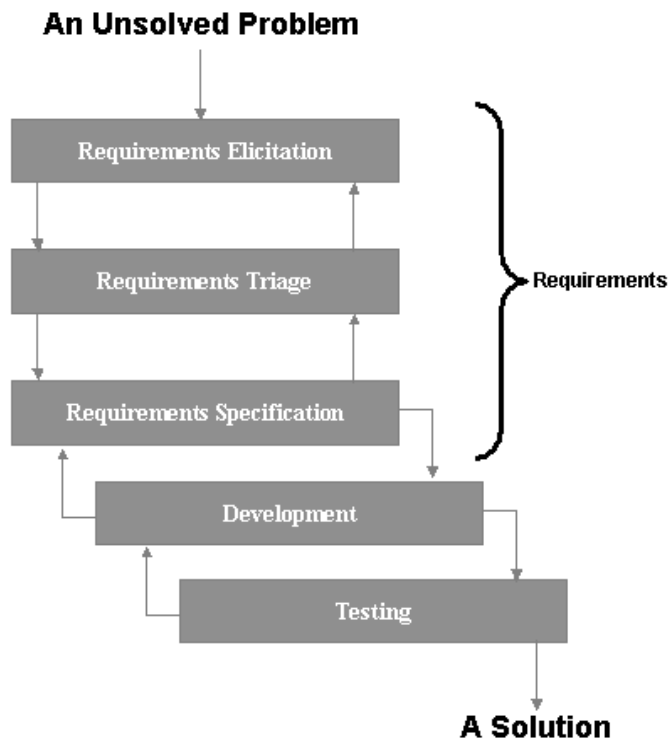
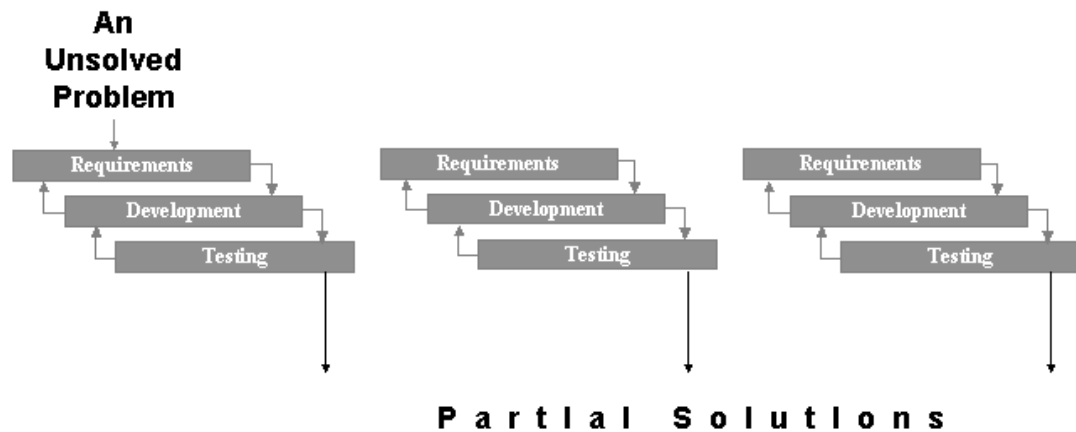


FIGURE 1.9. AN ITERATIVE DEVELOPMENT MODEL.

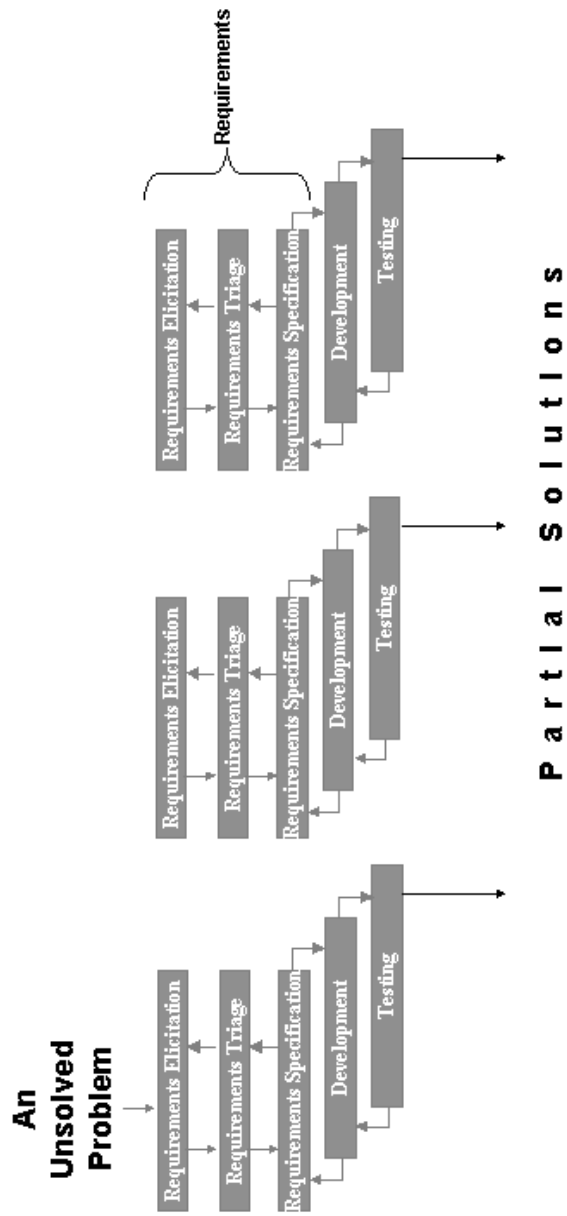
opment recommends that systems be built immediately. Agile developers construct iterations of the system in rapid succession, even as short as every day. A customer is required to be on-site with the development team at all times. Thus, requirements elicitation is performed constantly and is based primarily on the stimulation resulting from seeing system iterations. The omnipresent customer also has exclusive authority to select which requirements to include in each iteration. Thus, elicitation and triage are performed constantly, and specification is not performed *per se*.

Agile development is a reaction by software developers to what they perceive as too much control. The fact is that software development *is* difficult, and it requires a great deal of coordination. Agile development is likely to work well in situations where (a) the requirements are not changing, (b) there is only one customer (or there are more than one customer, but no conflicts exist among the stakeholders), (c) the problem is relatively simple, so that few misunderstandings concerning requirements are likely to arise.

Maintenance Projects

Once a system is deployed, the life of the system, in the eyes of the user, has just begun. Now that the user has had an opportunity to put the product through its paces, there will likely be plenty of feedback regarding the software. This feedback falls into two general categories: (a) failures of the product to meet the intended requirements and (b) requests for new features. The demand for new features will accelerate in any system that is being used (Belady and Lehman, 1976). Rather than allowing the system to be under constant flux, system evolution should be managed as a series of well-planned releases. The length of time between subsequent releases is a function of (a) the rate of arrival of new requirements, (b)

FIGURE 1.10. REQUIREMENTS ACTIVITIES WITHIN AN ITERATIVE DEVELOPMENT MODEL.



the overhead involved in producing and maintaining a release, and (c) the demand for early satisfaction. As each new requirement is discovered, it should be annotated just like the original requirements and documented in the same way that all previously approved requirements were. When the time arrives to initiate development of a new release, a triage meeting should be held. In principle, the management of post-deployment maintenance releases is no different than the management of predeployment iterations.

After a requirement is approved for a new release, multidirectional traces should be maintained between the change request, the new requirement, and all changes to the product and its documentation made in response to the change request. This enables the development team to (a) undo the changes if they prove erroneous and (b) reconstruct the history of changes made to the product.

Even with the best of processes in place, a product's entropy increases as it evolves (Lehman, 1978). The length of time that a system can survive is a function of the resiliency of the original architecture and the number of changes made over time. shows how the same system could last 7, 12, or 18 years before its entropy renders it no longer maintainable, based solely on the quality of the original architecture.

System Procurement

Many projects are commissioned to solve a problem by procuring, or acquiring, an available system from a third party. In such cases, requirements should still be elicited as described earlier. However, rather than performing an explicit triage step, the team generally prioritizes the elicited requirements and performs a "best fit" analysis with the available solutions.

Tool Issues

A requirements tool is a software application designed to assist the team in performing some combination of requirements elicitation, triage, and specification. Here is a list of the kinds of things such tools could do:

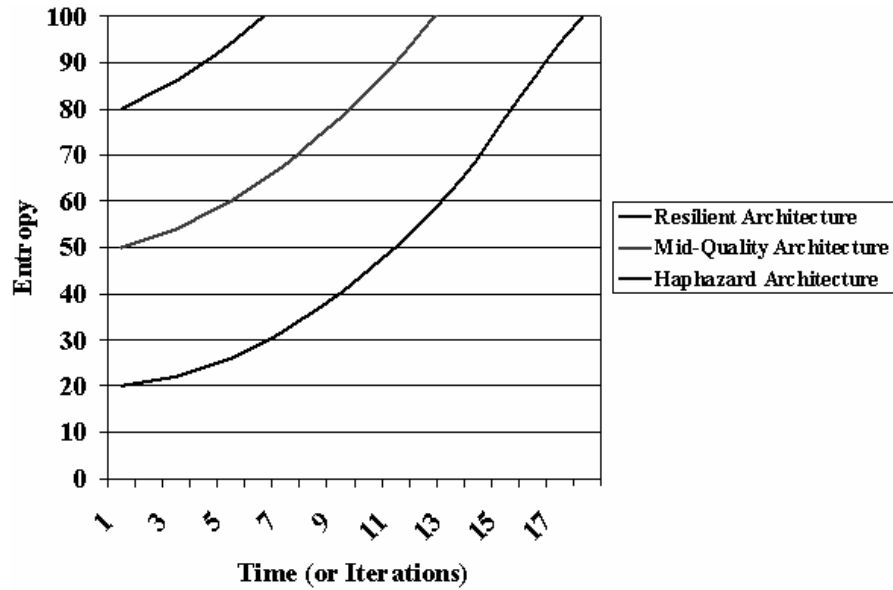
Elicitation

- Collect candidate requirements.
- Allow analysts to record lists of requirements as they are ascertained.
- Allow stakeholders to record their recommended requirements.
- Enforce discipline and/or protocol during elicitation sessions.
- Provide for anonymity during elicitation.
- Prompt for key missing information.

Triage

- Collect priorities and effort estimations.
- Allow analyst to record inclusion/exclusion of each requirement.

FIGURE 1.11. LONGEVITY OF A PRODUCT IS A FUNCTION OF ORIGINAL ARCHITECTURE'S RESILIENCY.



- Determine probability of completing a set of requirements within a given budget.
- Determine probability of completing a set of requirements within a given schedule.
- Allow analyst to refine requirements.

Specification

- Store requirements in a database.
- Determine ambiguities.
- Determine inconsistencies.
- Allow analyst to sort requirements based on multiple criteria.
- Allow analyst to cross-reference² requirements among themselves.
- Allow analyst to cross-reference² requirements to other products of the development effort (e.g., tests, designs).
- Provide the stakeholders with a simulation of the requirements (i.e., a prototype of the system).

²Also termed “traceability.”

Requirements tools range from such basic tools as spreadsheets and word processors to extremely sophisticated tools such as special-purpose requirements-based simulation tools. In general, they fall into the following categories:

- *General-purpose tools that happen to be useful during requirements activities.* Word processors allow you to record requirements in natural language either in paragraph form or tabular form. Spreadsheets and databases provide the same capability but also give you the ability to easily define and record attributes such as effort, priority, and inclusion easily. Examples of these tools are Microsoft Word or any other word processor, Microsoft Excel or any other spreadsheet, and Microsoft Access or any other database.

A majority of projects use these low-cost tools because they are already readily available on desktops with no additional cost. They also present no learning curve for the analysts, stakeholders, or project managers.

- *Meeting facilitation tools.* These tools are particularly helpful during elicitation. They enable stakeholders to record their suggested requirements easily, and even anonymously. They help to keep the discussion on-topic, can sort and filter the candidate requirements easily, and in some cases, can populate a requirements database tool. Two examples of such tools are Ventana's GroupSystems and Meetingworks' Connect.

Facilitation tools have had surprisingly little impact on most companies. Analysts performing elicitation tend to either interview stakeholders or hold group sessions without tools.

- *Requirements database and traceability tools.* These tools include a database view that is already populated with common requirements attributes. They provide special sorting and filtering capabilities unique to requirements management. Many also provide a word-processed view, so you can update requirements in either the word-processed view or the database view and the other updates automatically. Furthermore, all of these tools make cross-referencing and establishing relationships among requirements easy. Some of these tools are integrated into a full development environment, thus facilitating referencing to and from requirements, designs, and tests. Examples of these tools include RequisitePro from IBM Rational Software, Caliber RM from Borland Software Corporation, and DOORS from Telelogic.

Approximately 25 percent of all software development projects use requirements database and traceability tools. They significantly reduce the effort expended by analysts in recording and maintaining requirements, but have little impact directly on the stakeholders. One of their biggest advantages is to the project manager who can make intelligent and useful queries such as "Which requirements are high priority, included in the next release, and which are related to software components that Sally is working on."

- *Requirements risk analysis tools.* These tools help the project manager assess the likelihood that the selected requirements will be completed on schedule and within budget. Examples include OnYourMark Pro from Davis and the EstimatePro from Software Productivity Solutions, and part of Caliber RM from Borland Software Corporation.

These tools have been in existence only since the late 1990s. Early adopters have started experimenting with them, but their adoption has been slow. The primary benefit factor is the project manager and, indirectly, the company.

- *Requirements simulation tools.* These tools allow the requirements analyst to simulate the requirements after they have been written. In all cases, the requirements must first be written in a relatively formal notation. One example is Statemate Magnum from I-Logix.

These tools have been in existence since the early 1970s. All of the vendors have had a hard time finding their niche. The primary benefactor of such tools appears to be the engineering analyst.

In summary, requirements tools can assist analysts in all aspects of requirements management. But no tool makes any aspect of requirements management easy. Elicitation still requires great listening skills. Triage still requires great diplomacy, and specification still requires incredible precision. The tools simply offload the more mundane aspects of the discipline.

Trends in Requirements Management

Research

The field of requirements research is one of the most active in universities. Recent research surveys (Finkelstein, 1994; Hsia et al., 1993; van Lamsweerde et al., 2000; Nuseibeh et al., 2000; and Potts, 1991) have defined the following trends:

- *Data and process modeling* is viewed as a critical activity in requirements. Much of the research since the 1970s has focused on the creation and analysis of modeling notations and techniques. Two somewhat contradictory trends occurring in this area include (1) the increasing emphasis on object-oriented modeling notations (e.g., UML) that focus on the system and (2) the recognition that modeling cannot focus on the system in isolation but must occur in an organizational context (Nuseibeh et al., 2000; Goguen and Jirotko, 1994; and Zave and Jackson, 1997). More recent emphasis has been on techniques to detect errors in models. See the special issue of the *Requirements Engineering Journal* guest edited by Easterbrook and Chechik (2002).
- *Increasing formality* to improve the quality and testability of requirements specifications has been a goal of requirements research (Hsia et al., 1993), especially for process control and life- and safety-critical systems (van Lamsweerde et al., 2000). For example, in the area of reactive systems for process control, specification notations and languages such as SCR Heninger, 1980), CORE (Faulk, 1992), and RSML (Leveson et al., 1994) have been developed to support automated consistency and completeness checking. Formal specification languages such as Z (Spivey, 1990) and others are designed to support requirements verification, visualization, and simulation.
- *Viewpoints* explicitly capture different perspectives or views of multiple stakeholders. Viewpoint integration can be used to check for consistency and aid in the resolution of conflicts among stakeholders (Easterbrook, 1994; Nuseibeh and Easterbrook, 1994). The earliest references to using viewpoints date back to 1981 (Orr, 1981).
- Since the beginning of requirements research, attempts have been made to *reduce ambiguity* in requirements. Obviously, the aforementioned activities of modeling and increasing

formality are aimed at this goal. Additional research has been done to either reduce or detect ambiguity in natural-language specifications. This includes work as early as 1981 (Casey and Taylor, 1981) and extends to the current day (Duran et al., 2002).

- *Goal-oriented requirements elicitation* takes an organizational approach to completeness and consistency checking of requirements by explicitly identifying and representing organizational goals for the system, and then checking the requirements against those goals (van Lamsweerde et al., 2000). Research in this area has resulted in a variety of methods and notations for representing, analyzing, and resolving conflicts between goals including KAOS (Dardenne et al., 1993; van Lamsweerde et al., 1998) and NFR (Mylopoulos, 1992).
- Behavioral requirements have always been the primary emphasis in requirements research. However, *nonbehavioral requirements* have also been addressed for many years and continues to be the focus of many research efforts. Some efforts have spanned the wide range of nonbehavioral requirements, for instance Chung et al. (1993), Chung (2000), Cysneiros and Leite (2002), Kirner and Davis (1995) Mostert and van Solms (1995), and Mylopoulos (1992), and others emphasize specific kinds of nonbehavioral requirements such as security (Shim and Shim, 1992), safety (Berry, 1998; Hansen et al., 1998), and performance (Nixon, 1993).
- *Scenarios* are concrete descriptions of the sequence of activities that users engage in when performing a specific task (Carroll, 1995). Studies have shown that scenarios are extremely useful for requirements elicitation when users are having difficulty specifying goals or using more abstract modeling techniques (Weidenhaupt, 1998; Jarke, 1999; van Lamsweerde, 2000). Scenarios have also proven useful in systems design and testing, for example, in user interface design (Carroll, 1995), and for generating test cases (Hsia, 1994). Other scenario uses are described in an *IEEE Transactions on Software Engineering* special issue on scenarios in (Jarke and Kurki-Suonio, 1998). Finally, scenarios are closely related to the Jacobson's use cases (Jacobson et al., 1992) in object-oriented analysis and the user stories, which are a key component of XP (Beck, 2000).
- With the wide variety of requirements techniques now in existence, some researchers are focusing on the *criteria for technique selection*. For example, Hickey and Davis (2003, 2003a) describe the best way to select the right elicitation techniques. Similar research still needs to be conducted for model selection.
- The field of software (design and code) reuse has settled into a status quo now; modern programming languages include large libraries of reusable entities whose use has become standard. However, *requirements reuse* has not yet reached this level of maturity. Perhaps this is because reusing requirements has little direct benefit to increasing quality or productivity. Instead, the real potential benefit of requirements reuse comes from the second-order effect of reusing design and code components associated with the reused requirements. See Castano and Antenellis (1993), Homod and Rine (1999), van Lamsweerde (1997), and Maiden and Sutcliffe (1996) for some of the latest ideas on requirements reuse.

Practice

It is surprising how little of the current research in the requirements field is making its way to practice (Davis and Hickey, 2002). From the inception of software engineering as a

discipline in the 1970s until the current day, (a) the standard for documenting requirements has been the word-processed SRS, (b) analysts in specialized applications have advocated the use of models, and (c) a counterculture has existed that is firmly convinced that writing requirements is primarily a waste of time.

In spite of the enormity of these invariants, a few changes have occurred. Two of these changes are in the evolution of the modeling notations themselves. The first is the introduction of new notations that provide unique perspectives of the system under specification. Classic among these are the introductions of statecharts by Harel (Harel, 1988; and Harel and Politi, 1998). Second is the tendency for the industry to move from sets of specialized notations (which in theory force analysts to become skilled in multiple languages) to all-encompassing notations (which in theory force analysts to become skilled in just one language, albeit enormous), and back to the specialized languages in a cycle. We expect this cycle to continue indefinitely into the future.

Another trend is in the isolation of optimal “starting points” for requirements activities. For many years, analysts have struggled with the question of where to start because of the sheer enormity of requirements. We have thus seen structured analysis (DeMarco, 1979) augmented by events as starting points (McMenamin and Palmer, 1984), and object-oriented analysis (Booch, 1994) augmented with use cases as starting points (Jacobson et al., 1992). This trend will continue. Unfortunately, every situation demands starting points that are a unique function of situational characteristics.

Summary

Project management cannot succeed without careful attention to requirements management. Requirements management is responsible for determining the real needs of the customers, as well as clearly documenting the desired external behavior of the system being constructed by the project. If either of these goals is ignored, the project is guaranteed to result in failure.

References

- Beck, K. 2000. *Extreme programming explained*. Boston: Addison-Wesley.
- Belady, L., and M. Lehman. 1976. A model of large program development. *IBM Systems Journal* 15 (3, March): 225–252.
- Berry, D. 1998. The safety requirements engineering dilemma. *Ninth International Workshop on Software Specification and Design*. 147–149. Los Alamitos, CA: IEEE Computer Society Press.
- Boehm, B. 1982. *Software engineering economics*. Upper Saddle River, NJ: Prentice Hall.
- Booch, G., 1994. *Object-oriented analysis and design*. Redwood City, CA: Benjamin/Cummings.
- . Personal conversation with two of the authors; September 17, 2002, Colorado Springs, Colorado.
- Booch, G., et al. 1999. *The Unified Modeling Language user guide*. Reading, MA: Addison-Wesley.
- Borland Software Corporation, Inc. 2003. See www.borland.com/products or www.starbase.com/products.
- Carroll, J., ed. 1995. *Scenario-based design: Envisioning work and technology in system development*. New York: Wiley.

- Casey, B., and B. Taylor. 1981. Writing requirements in English: A natural alternative. 95–101. *IEEE Software Engineering Standards Workshop*. Los Alamitos, CA: IEEE Computer Society Press.
- Castano, S., and V. De Antonellis. 1993. Reuse of conceptual requirements specification. 121–124. *International Symposium on Requirements Engineering*, January. Los Alamitos, CA: IEEE Computer Society Press.
- Christensen, M., and R. Thayer. 2001. *The project manager's guide to software engineering's best practices*. Los Alamitos, CA: IEEE Computer Society Press.
- Chung, L. 1993. *Representing and using non-functional requirements: A process-oriented approach*. Department of Computer Science. PhD. thesis, University of Toronto.
- Chung, L., et al. 2000. *Non-functional requirements in software engineering*. Norwell, MA: Kluwer.
- Cleland, D., and L. Ireland. 2000. *Project manager's portable handbook*. New York: McGraw-Hill.
- Cockburn, A. 2002. *Agile software development*. Boston: Addison-Wesley.
- Cysneiros, M., and J. Leite. 2002. Non-functional requirements: From elicitation to modeling languages. 699–700. *Twenty-fourth International Conference on Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press.
- Dardenne, A., et al. 1993. Goal-directed requirements acquisition. *Science of Computer Programming* 20:3–50.
- Davis, A., 1993. *Software requirements: Objects, functions, and states*. Upper Saddle River, NJ: Prentice Hall.
- . 1995. Software prototyping. *Advances in Computers* 40. 39–63. New York: Academic Press.
- . 2002. Requirements management. In *Encyclopedia of software engineering*, 2nd ed., ed. J. Marciniak. New York: Wiley-Interscience.
- . 2003. Secrets of requirements triage. *Computer* 36 (3, March): 42–49.
- Davis, A., and A. Zweig. 2000. The missing piece of software development. *Journal of Systems and Software* 53 (3, September): 205–206.
- Davis, A., et al. 1993. Identifying and measuring quality in software requirements specifications. 141–152. *IEEE-CS International Software Metrics Symposium*. Los Alamitos, CA: IEEE Computer Society Press.
- Davis, A., and A. Hickey. 2002. Requirements researchers: Do we practice what we preach? *Requirements Engineering Journal* 7(2):107–111.
- Dean, D., et al. (1997–1998). Enabling the effective involvement of multiple users: Methods and tools for collaborative software engineering. *Journal of Management Information Systems* 14 (3, Winter): 179–222.
- DeMarco, T. 1979. *Structured analysis and system specification*. Upper Saddle River, NJ: Prentice Hall.
- Duran, A., et al. 2002. Verifying software requirements with XSLT. *ACM Software Engineering Notes* 27: 39 ff.
- Easterbrook, S. 1994. Resolving requirements conflicts with computer-supported negotiation. In *Requirements engineering: Social and technical Issues*, ed. M. Jirotko and J. Goguen. 41–65. London: Academic Press.
- Easterbrook, S., and M. Chechik. 2002. Guest editorial: Special issue on model checking in requirements engineering. *Requirements Engineering* 7(4):221–224.
- Faulk, S. 1997. Software requirements: A tutorial. In *Software Requirements Engineering*, ed. R. Thayer and M. Dorfman. 128–149. Los Alamitos, CA: IEEE Computer Society.
- Faulk, S., et al. 1992. The CORE method for real-time requirements *IEEE Software* (September): 22–33.
- Finkelstein, A. 1994. Requirements engineering: A review and research agenda. 10–14. *First Asia-Pacific Software Engineering Conference*. December. Los Alamitos, CA: IEEE Computer Society.
- Gause, D., and J. Weinberg. 1989. *Exploring requirements: Quality before design*. New York: Dorset House.
- Goguen, J., and C. Linde. 1993. Software requirements analysis and specification in Europe: An overview. 152–164. *First International Symposium on Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press.

- Goguen, J., and M. Jirotko, eds. 1994. *Requirements engineering: Social and technical issues*. Boston: Academic Press.
- Gottesdeiner, E. 2002. *Requirements by collaboration*. Reading, MA: Addison-Wesley.
- Hansen, K., et al. 1998. From safety analysis to software requirements. *IEEE Transactions on Software Engineering* 24 (7, July): 573–584.
- Harel, D. 1988. On visual formalisms. *Communications of the ACM* 31 (5, May): 514–530.
- Harel, D., and M. Politi 1998. *Modeling reactive systems with statecharts*. New York: McGraw Hill.
- Heninger, K. 1980. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering* 6(1):2–13.
- Hickey, A., and A. Davis. 2002. The role of requirements elicitation techniques in achieving software quality. *International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ)*. Los Alamitos, CA: IEEE Computer Society Press.
- . 2003a. Requirements elicitation and requirements elicitation technique selection: A model of two knowledge-intensive software development processes. *Proceedings of the Thirty-Sixth Hawaii International Conference on System Sciences*. Los Alamitos, CA: IEEE Computer Society Press.
- . 2003b. Elicitation technique selection: How do the experts do it? International Joint Conference on Requirements Engineering (RE03). September. Los Alamitos, CA: IEEE Computer Society Press.
- Highsmith, J., and A. Cockburn. 2001. Agile software development: The business of innovation. *Computer* (September): 120–122.
- Hofmann, H., and F. Lehner 2001. Requirements engineering as a success factor in software projects. *IEEE Software* 18 (4, July/August): 58–66.
- Homod, S., and D. Rine. 1999. Building requirements repository using requirements transformation techniques to support requirements reuse. *World Multi-Conference on Systemics, Cybernetics and Informatics*, Volume 2.
- Hsia, P., et al. 1993. Status report: Requirements engineering. *IEEE Software* 10 (6, November): 75–79.
- Hsia, P., et al. 1994. Formal approach to scenario analysis. *IEEE Software* 11(2):33–41.
- IEEE. 1983. *IEEE standard glossary of software engineering terminology*. IEEE Standard 729. New York: IEEE Press.
- . 1986. *IEEE standard for software verification and validation plans*. IEEE Standard 1012. New York: IEEE Press.
- . 1993. *A guide to software requirements specifications*. Standard 830-1993. New York: IEEE Press.
- I-Logix Corporation. www.ilogix.com/products/magnum/index.cfm.
- Jacobson, I., et al. 1992. *Object-oriented software engineering*. Reading, MA: Addison-Wesley.
- Jarke, M., and R. Kurki-Suonio. 1998. Guest editorial: Introduction to the special issue. *IEEE Transactions on Software Engineering* 24(12):1033–1035.
- Jarke, M. 1999. Scenarios for modeling. *Communications of the ACM* 42(1): 47–48.
- Kirner, T., and A. Davis. 1996. Nonfunctional requirements for real-time systems. *Advances in Computers*.
- Knapp, M., and J. Hall. 1997. *Nonverbal communication in human interaction*. Austin, TX: Holt, Rinehart and Winston.
- Kotonya, G., and I. Sommerville. 1997. Integrating safety analysis and requirements engineering. 259–271. *Fourth Asia-Pacific Software Engineering Conference*. Los Alamitos, CA: IEEE Computer Society.
- Kowal, J. 1992. *Behavior models*. Upper Saddle River, NJ: Prentice Hall.
- Lam, W., et al. 1997. Ten steps towards systematic requirements reuse. 6–15. *IEEE International Symposium on Requirements Engineering*. January Los Alamitos, CA: IEEE Computer Society Press. Also appears in *Requirements Engineering Journal* 2(2):102–113.
- van Lamsweerde, A. 2000. Requirements engineering in the year 00: A research perspective. *Proceedings of the 22nd International Conference on Software Engineering*. 5–19. New York: ACM Press.

- van Lamsweerde, A., et al. 1998. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering* 24 (11, November): 908–926.
- Lauesen, S. 2002. *Software requirements: Styles and techniques*. London: Addison-Wesley.
- Leffingwell, D., and D. Widrig. 2000. *Managing software requirements*. Reading, MA: Addison-Wesley.
- Lehman, M. 1978. *InfoTech State of the Art Conference on Why Software Projects Fail*. Paper #11, April.
- Leveson, N., et al. 1994. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering* 20 (9, September): 684–706.
- McMenamin, J., and J. Palmer. 1984. *Essential systems analysis*. Upper Saddle River, NJ: Prentice Hall.
- Maiden, N., and A. Sutcliffe. 1996. Analogical retrieval in reuse-oriented requirement engineering. *Software Engineering Journal* 11(5):281–292.
- Meetingworks, Inc. 2003. www.meetingworks.com.
- Meredith, J., and S. Mantel. 2003. *Project management: A managerial approach*. 5th ed. New York: Wiley.
- Microsoft, Inc. 2003. www.microsoft.com.
- Mostert, D., and S. von Solms. 1995. A technique to include computer security, safety, and resilience requirements as part of the requirements specification. *Journal of Systems and Software* 31 (1, October): 45–53.
- Mylopoulos, J., et al. 1992. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering* 18(6, June): 483–497.
- Nixon, B. 1993. Dealing with performance requirements during the development of information systems. 42–49. *IEEE International Symposium on Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press.
- Nuseibeh, B., et al. 1994. A framework for expressing the relationships between multiple views in requirements specifications. *IEEE Transactions on Software Engineering* 20 (10, October): 760–773.
- Nuseibeh, B., and S. Easterbrook. 2000. Requirements engineering: A roadmap. *Proceedings of the 22nd International Conference on Software Engineering*. 35–46. New York: ACM Press.
- Opdahl, A. 1994. Requirements engineering for software performance, *International Workshop on Requirements Engineering: Foundations of Software Quality*. June.
- Orr, K. 1981. *Structured requirements definition*. Topeka, Kansas: Ken Orr and Associates.
- Project Management Institute. 2000. *A guide to the project management body of knowledge*. Newtown Square, PA: Project Management Institute.
- Potts, C. 1991. Seven (plus or minus two) challenges for requirements research. *Sixth International Workshop on Software Specification and Design*. Los Alamitos, CA: IEEE Computer Society.
- Rational Software Corporation, Inc. 2003. www.rational.com/products.
- Robertson, J., and S. Robertson. 2000. *Mastering the requirements process*. Reading, MA: Addison-Wesley.
- Reinertsen, D. 1997. *Managing the design factory*. New York: Free Press.
- Shim, Y., H. Shim, et al. 1997. Specification and analysis of security requirements for distributed applications. 374–381. *Ninth IEEE International Conference on Software Engineering and Knowledge Engineering*. June. Skokie, IL: Knowledge Systems Institute.
- Siddiqi, J., and C. Shekaran. 1996. Requirements engineering: The emerging wisdom. *IEEE Software* 13(2):15–19.
- Software Productivity Centre, Inc. 2003. <http://www.spc.ca/products/estimate>.
- Spivey, J. 1990. An introduction to Z and formal specifications. *Software Engineering Journal* 4:40–50.
- The Standish Group. Undated. *The CHAOS Chronicles* www.standishgroup.com.
- Swartout, W., and R. Balzer 1982. On the inevitable intertwining of specifications and design. *Communications of the ACM* 25 (7, July): 438–440.
- Telelogic, Inc. 2003. www.telelogic.com/products.
- Thayer, R., and M. Dorfman 1994. *Standards, guidelines, and examples on system and software requirements engineering*. Los Alamitos, CA: IEEE Computer Society Press.

- Ventana, Inc. 2003. www.ventana.com.
- Wallace, D. 1994. Verification and validation. In *Encyclopedia of Software Engineering*, ed., J. Marciniak. 1410–1433. New York: Wiley.
- Weidenhaupt, K., et al. 1998. Scenarios in system development: Current practice. *IEEE Software* 15(2): 34–45.
- Wieringa, R. 1996. *Requirements engineering*. Chichester, UK: Wiley.
- Wiegers, K. 1999. *Software requirements*. Redmond, WA: Microsoft Press.
- Wood, J., and D. Silver 1995. *Joint application development*. 2nd ed. New York: Wiley.
- Young, R. 2001. *Effective requirements practices*. Boston: Addison-Wesley.
- Zave, P., and M. Jackson. 1997. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology* 6 (1, January): 1–30.