Chapter 9

R Management

Requirements Management

Introduction

Requirements management involves identifying, documenting, and tracking system requirements from inception through delivery. Inherent in this definition is understanding of the true meaning of the requirements and the management of customer (and stakeholder) expectations throughout the system's life cycle. A solid requirements management process is the key to a successful project.

Hull, Jackson, and Dick (2011) suggest there are five main challenges to requirements management. The first is that very few organizations have a well-defined requirements management process, and thus, few people in those organizations have requirements management experience. The second challenge is the difficulty in distinguishing between user or stakeholder requirements and systems. The third problem is that organizations manage requirements differently, making the dissemination and transferability of experience and best practices difficult. The difficulties in progress monitoring are yet another problem. And finally, they suggest that managing changing requirements is a significant challenge. They suggest that establishing a well-defined requirements management process is key in addressing these issues.

Reinhart and Meis (2012) discuss how certain requirements management approaches adapted from production engineering can be used to improve the execution of "simultaneous engineering," that is, the parallelization of unequal activities that are generally conducted sequentially. They suggest that success factors include simulation prototyping, integration of available knowledge, frequent communication, and efficient use of tools.

Most organizations do not have an explicit requirements management process in place, but this does not mean that requirements management does not occur within the organization. The requirements practices probably exist implicitly in the organization, but these practices are not usually documented. One of the first steps in improving the requirements management process in any organization is to document existing practices.

Reconciling Differences

One of the most important activities in project management is consensus building, particularly when forming or discovering:

Mission statements
Goals
Requirements
Rankings

But achieving consensus between stakeholder groups is not easy. Tuckman's theory of team formation suggests that group formation follows a pattern of "Forming, Storming, Norming, Performing, and Adjourning." The premise is that a team can dramatically change from a noncohesive group into a high-functioning one. Interested readers might want to watch Sidney Lumet's 1957 movie, *12 Angry Men*. You can actually see the jury going through the Tuckman team formation sequence, and it is also a great film (Neill et al. 2012).

Managing Divergent Agendas

Each stakeholder has a different requirements "agenda." For example, business owners seek ways to get their money's worth from projects. Business partners want explicit requirements because they are like a "contract." Senior management expects more financial gain from projects than can be realized. And systems and software developers like uncertainty because it gives them freedom to innovate solutions. Project managers may use the requirements to protect them from false accusations of underperformance in the delivered product.

One way to understand why the existence of different agendas—even among persons within the same stakeholder group—is the "Rashomon Effect." *Rashomon* is a highly revered 1950 Japanese film directed by Akira Kurosawa. The main plot involves the recounting of the murder of a samurai from the perspective of four witnesses to that event: the samurai (through a medium), his wife, a bandit, and a wood cutter, each of whom has a hidden agenda, and tells a contradicting accounting of the event. Stated succinctly, "Your understanding of an event is influenced by many factors, such as your point of view and your interests in the outcome of the event" (Lawrence 1996).

The smart requirements manager seeks to manage these agendas by asking the right questions up front. Andriole (1998) suggests the following questions are appropriate:

1. What is the project request?
   Who wants it?
   Is it "discretionary" or "nondiscretionary"?
2. What is the project's purpose?
   If completed, what impact will the new or enhanced system have on organizational performance?
   On profitability?
   On product development?
   On customer retention and customer service?
3. What are the functional requirements?
   What are the specific things the system should do to satisfy the purposeful requirements?
   How should they be ranked?
   What are the implementation risks?
4. What are the nonfunctional requirements, such as security, usability, and interoperability?
   How should they be ranked?
   What are the implementation risks?
   How do you trade off functional and nonfunctional requirements?
5. Do we understand the project well enough to prototype its functionality?
6. If the prototype is acceptable, will everyone sign off on the prioritized functionality and nonfunctionality to be delivered, on the initial cost and schedule estimates, on the estimates' inherent uncertainty, on the project's scope, and on the management of additional requirements?

Andriole asserts that by asking these questions up front, hidden agendas can be uncovered and differences resolved. At the very least, important issues will be raised up front and not much later in the process.

Consensus Building

There are numerous approaches to consensus building including:

Negotiation (argument)
Executive fiat
Mathematical techniques, including data fusion, and the analytical hierarchy process (AHP)
Appeals to an expert

But perhaps the most celebrated consensus building technique in software and systems engineering is the Wideband Delphi technique. Developed in the 1970s, but popularized by Barry Boehm in the early 1980s, Wideband Delphi is usually associated with the selection and prioritization of alternatives. These alternatives are usually posed in the form of a question:

> For the following alternative, rate your preference according to the following scale (5 = most desired, 4 = desired, 3 = ambivalent, 2 = not desired, 1 = least desired).

The list of alternatives and associated scale is presented to a panel of "experts" (and in the case of requirements ranking, stakeholders) who rank these requirements silently and sometimes anonymously. The ranking scale can have any number of levels. The collected list of rankings is then re-presented to the group by a coordinator. Usually, there is significant disagreement in the rankings. A discussion is conducted and experts are asked to justify their differences in opinion. After discussion, the independent ranking process is repeated (see Figure 9.1).

With each iteration, individual customer rankings should start to converge and the process continues until a satisfactory level of convergence is achieved.

A more structured form of the process is:

1. Coordinator presents each expert with a specification and an estimation form.
2. Coordinator calls for a group meeting in which the experts discuss estimation issues with the coordinator and each other.
3. Experts fill out forms anonymously.
4. Coordinator prepares and distributes a summary of the estimates.

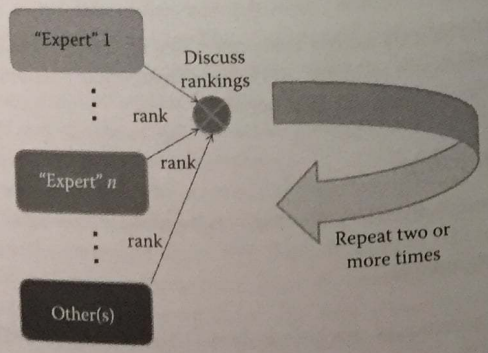

Figure 9.1 The Wideband Delphi process.

5. Coordinator calls for a group meeting, specifically focusing on having the experts discuss points where their estimates vary widely.
6. Experts fill out forms, again anonymously, and Steps 4 to 6 are iterated for as many rounds as appropriate.

There is usually not unanimous agreement in the Wideband Delphi process, but at least everyone involved will feel that his or her opinion has been considered. Wideband Delphi is a kind of win–win negotiating and can be used for other types of decision making.

Expectation Revisited: Pascal's Wager

Mathematician Blaise Pascal (1623–1662) is well known for various achievements including Pascal's triangle (a convenient way to find binomial coefficients) and work in probability. It was his work in probability theory that led to the notion of expected value, and he used such an approach later in life when he became more interested in religion than mathematics. It was during his monastic living that he developed a theory that suggested that it was advisable to live a virtuous life, whether or not one believed in a Supreme Being. His approach, using expected value theory, is now called Pascal's wager, and it goes like this.

Imagine an individual is having a trial of faith and is unsure if he believes in this Supreme Being (let's call him "God" for argument's sake). Pascal suggests that it is valuable to consider the consequences of living virtuously, in the face of the eventual realization that such a God exists (or not). To see this, consider Table 9.1.

Assuming that it is equally probable that God exists or not (this is a big assumption), we see that the expected outcome (consequence) of living virtuously is half of Paradise and the expected outcome of living without virtue is half of Damnation. Therefore, it is in a person's best interests to live virtuously.

What does Pascal's wager have to do with expectation setting? Stakeholders will hedge their bets, sometimes withholding information or offering up inferior information because they are playing the odds involving various organizational issues. For example, does a stakeholder wish to request a feature that she believes no one else wants and for which she might be ridiculed? From a game theory standpoint it

Table 9.1 Pascal's Wager Consequence Matrix

	God Exists	*God Does Not Exist*
Live virtuously	Achieve paradise	Null
Do not live virtuously	Achieve damnation	Null

Table 9.2 Modified Pascal's Wager Consequence Matrix

	Group Agrees	*Group Disagrees*
Speak out	Get praise	Get ridiculed
Remain silent	Nothing happens	Nothing happens

is safer for her to withhold her opinion. To see this, consider the modified Pascal's wager outcome matrix in Table 9.2.

Suppose the stakeholder decides to speak out about a particular feature (or in opposition to a particular feature). The consequence matrix shows that she can expect to get some praise if the group agrees or some ridicule if the group disagrees, assuming equi-likely probability that the group will agree or disagree, and equal praise and ridicule, the "payoff" is equivalent to saying nothing. It is well known that, in decision making, individuals will tend to make decisions that avoid loss over those that have the potential for gain; most individuals are risk averse. But the expected consequences are much worse if the stakeholder believes there is a strong chance that her colleagues will disagree. Therefore, her logical course of action is to say nothing. Of course, later in the process the feature might suddenly be discovered by others to be important. Now it is very late in the game, however, and adding this feature is costly. Had the stakeholder only spoken up in the beginning, in a safe environment for discussion, a great deal of cost and trouble could have been avoided.

One last comment on Pascal's wager, expectation, and risk. The author once worked for a boss—we'll call him "Bob"—who greeted all new employees with a welcome lunch. At that lunch he would declare "I am a 'no surprises' kind of guy. You don't surprise me, and I won't surprise you. So, if there is ever anything on your mind, or any problem brewing, I want you to bring that to my attention right away." This sentiment sounded great. However, each time the author or anyone else would bring bad news to Bob, whether the messenger was responsible for the situation or not, Bob would blow his stack and berate the hapless do-gooder. After a while, potential messengers would forgo bringing information to Bob. The decision was purely game theoretic: if you had bad news and you brought it to Bob, he would yell at you. If he didn't find out about it (as often happened because no one else would tell him, either), then you escaped his rampage. If he somehow found out about the bad news, you might get yelled at, but he might yell at the first person in his sight, not you, even if you were the party responsible for the problem. So, it made sense (and it was rigorously sound via game theory) to shut up.

It was rather ironic that "no surprise Bob" was always surprised because everyone was afraid to tell him anything. The lesson here is that, if you "shoot the messenger," people will begin to realize the consequences of bringing you information, and you will soon be deprived of that information. Actively seeking and inviting requirements information throughout the project life cycle is an essential aspect of requirements management.

Global Requirements Management

Requirements engineering is one of the most collaboration-intensive activities in systems engineering. Because of inadequate social contact, geographically distributed stakeholders and REs have trouble understanding requirements issues without the appropriate tools. Agile development seeks to solve this problem for software through always on-site customers (Sinha and Sengupta 2006).

Global and even onshore outsourcing present all kinds of challenges to the requirements engineering endeavor. These include time delays and time zone issues, the costs and stresses of physical travel to client and vendor sites when needed, and the disadvantages of virtual conferencing and telephone. Even simple e-mail communications cannot be relied upon entirely, even though for many globally distributed projects informal e-mails and e-mail distributed documents are the main form of collaboration. But e-mail use leads to frequent context switching, information fragmentation, and the loss of nonverbal cues.

When the offshoring takes place in a country with a different native language and substantially different culture, new problems may arise in terms of work schedules, work attitudes, communication barriers, and customer–vendor expectations of how to conduct business. Moreover, offshoring introduces a new risk factor: geopolitical risk, and this risk must be understood, quantified, and somehow factored into the requirements engineering process and schedule. Finally, there are vast differences in laws, legal process, and even the expectations of honesty in business transactions around the world. These issues are particularly relevant during the requirements elicitation phase.

Bhat, Gupta, and Murthy (2006) highlighted nine specific problems they observed or experienced. These included:

- Conflicting client–vendor goals
- Low client involvement
- Conflicting requirements engineering approaches (between client and vendor)
- Misalignment of client commitment with project goals
- Disagreements in tool selection
- Communication issues
- Disowning responsibility
- Sign-off issues
- Tools misaligned with expectation

Bhat et al. suggest that the following success factors were missing in these cases, based on an analysis of their project experiences:

- Shared goal: that is, a project "metaphor"
- Shared culture: in the project sense, not in the sociological sense
- Shared process
- Shared responsibility
- Trust

These suggestions are largely consistent with agile methodologies, although we have already discussed the challenges and advantages of using agile approaches to requirements engineering.

Finally, what role can tools play in the globally distributed requirements engineering process? Sinha and Sengupta (2006) suggest that software tools can play an important role, although there are not many appropriate tools for this purpose. Appropriate software tools must support:

- Informal collaboration
- Change management
- Promoting awareness (e.g., auto-e-mail stakeholders when triggers occur)
- Managing knowledge (provide a framework for saving and associating unstructured project information)

There are several commercial and even open source solutions that claim to provide these features, but we leave the product research of these to the reader.

Antipatterns in Requirements Management*

In troubled organizations the main obstacle to success is frequently accurate problem identification. Diagnosing organizational dysfunction is quite important in dealing with the underlying problems that will lead to requirements engineering problems.

Conversely, when problems are correctly identified, they can almost always be dealt with appropriately. But organizational inertia frequently clouds the situation or makes it easier to do the wrong thing rather than the right thing. So how can you know what the right thing is if you've got the problem wrong?

In their groundbreaking book, Brown et al. (1998) described a taxonomy of problems or antipatterns that can occur in software architecture and design and in

* Some of this discussion is excerpted from Neill, Laplante, and DeFranco, *Antipatterns: Identification, Refactoring, and Management*, second edition. Boca Raton, FL: CRC Press (2012). With permission.

the management of software projects. They also described solutions or refactorings for these situations. The benefit of providing such a taxonomy is that it assists in the rapid and correct identification of problem situations, provides a playbook for addressing the problems, and provides some relief to the beleaguered employees in these situations in that they can take consolation in the fact that they are not alone.

These antipatterns bubble up from the individual manager through organizational dysfunction and can manifest in badly stated, incomplete, incorrect, or intentionally disruptive requirements.

Our antipattern set consists of an almost even split of 28 environmental (organizational) and 21 management antipatterns. *Management antipatterns* are caused by an individual manager or management team ("the management"). These antipatterns address issues in supervisors who lack the talent or temperament to lead a group, department, or organization. *Environmental antipatterns* are caused by a prevailing culture or social model. These antipatterns are the result of misguided corporate strategy or uncontrolled sociopolitical forces. But we choose to describe only a small subset of the antipattern set that is particularly applicable in requirements engineering.

Environmental Antipatterns

Divergent Goals

Everyone must pull in the same direction. There is no room for individual or hidden agendas that don't align with those of the business. The divergent goals antipattern exists when there are those who pull in different directions.

There are several direct and indirect problems with divergent goals:

Hidden and personal agendas divergent to the mission of an organization starve resources from strategically important tasks.

Organizations become fractured as cliques form to promote their own self-interests.

Decisions are second-guessed and subject to "review by the replay official" as staff try to decipher genuine motives for edicts and changes.

Strategic goals are hard enough to attain when everyone is working toward them; without complete support they become impossible and put risk to the organization.

There is a strong correspondence between stakeholder dissonance and divergent goals, so be very aware of the existence of both.

Because divergent goals can arise accidentally and intentionally there are two sets of solutions or refactorings. Dealing with the first problem of comprehension and communication involves explaining the impact of day-to-day decisions on larger objectives. This was the case with the corporate executives of the box

company described in Chapter 2. The executives forgot the bigger picture. There is more than providing a coffee mug with the mission statement, however. Remember that the misunderstanding is not because the staff are not aware of the mission or goals; organizations are generally very good at disseminating them. It is that they don't understand that their decisions have any impact on those goals. They have a very narrow perspective on the organization and that must be broadened.

The second problem of intentionally charting an opposing course is far more insidious, however, and requires considerable intervention and oversight. The starting point is to recognize the disconnect between their personal goals and those of the organization. Why do they feel that the stated goals are incorrect? If the motives really are personal, that they feel their personal success cannot come with success of the organization, radical changes are needed. Otherwise the best recourse is to get them to buy into the organizational goals. This is most easily achieved if every stakeholder is represented in the definition and dissemination of the core mission and goals, and subsequently kept informed, updated, and represented.

Process Clash

A process clash is the friction that can arise when advocates of different processes must work together without a proven hybrid process being defined. The dysfunction appears when organizations have two or more well-intended but noncomplementary processes; a great deal of discomfort can be created for those involved. Symptoms of this antipattern include poor communications—even hostility—high turnover, and low productivity.

The solution to a process clash is as follows: develop a hybridized approach, one that resolves the differences at the processes' interfaces. Retraining and cross-training could also be used. For example, by training the analysis group in XP and the development group in RUP, better understanding can be achieved.

Another solution is to change to a third process that resolves the conflict. For example, domain-driven modeling might have been used instead of RUP. Domain-driven modeling can be used in conjunction with agile methodologies with no conflicts.

There is a strong correlation between the aforementioned two antipatterns, and certain capability maturity models, such as the CMMI, can be used to help identify and reconcile process clashes that lead to divergent goals.

Management Antipatterns

Metric Abuse

The first management antipattern that might arise in requirements engineering is metric abuse, that is, the misuse of metrics either through incompetence or with deliberate malice (Dekkers and McQuaid 2002).

At the core of many process improvement efforts is the introduction of a measurement program. In fact sometimes the measurement program is the process improvement. That is to say, some people misunderstand the role measurement plays in management and misconstrue its mere presence as an improvement. This is not a correct assumption. When the data used in the metric are incorrect or the metric is measuring the wrong thing, the decisions made based upon them are likely the wrong ones and will do more harm than good.

Of course, the significant problems that can arise from metric abuse depend on the root of the problem: incompetence or malice. Incompetent metrics abuse arises from failing to understand the difference between causality and correlation, misinterpreting indirect measures, or underestimating the effect of a measurement program. Here's an example of the origin of such a problem. Suppose a fire control system for a factory is required to dispense fire retardant in the event of a fire. Fire can be detected in a number of ways: based on temperature, on the presence of smoke, the absence of oxygen on the presence of gasses from combustion, and so on. So, which of these should be measured to determine if there is a fire? Selecting the wrong metric can lead to a case of metrics abuse.

Malicious metrics abuse is derived from selecting metrics that support or decry a particular position based upon a personal agenda. For example, suppose a manager institutes a policy that tracks the number of requirements written per engineer per day and builds a compensation algorithm around this metric. Such an approach is simplistic and does not take into account the varying difficulties in eliciting, analyzing, agreeing, and writing different kinds of requirements. In fact, the policy may have been created entirely to single out an individual who may be working meticulously, but too slowly for the manager's preference.

The solution or refactoring for metrics abuse is to stop the offending measurements. Measuring nothing is better than measuring the wrong thing. When data are available, people use them in decision making, regardless of their accuracy.

Once the decks have been cleared, Dekkers and McQuaid (2002) suggest a number of steps necessary for the introduction of a meaningful measurement program.

1. *Define measurement objectives and plans*: Perhaps by applying the goal-question-metric (GQM) paradigm.
2. *Make measurement part of the process*: Don't treat it like another project that might get its budget cut or that one day you hope to complete.
3. *Gain a thorough understanding of measurement*: Be sure you understand direct and indirect metrics; causality versus correlation; and, most importantly, that metrics must be interpreted and acted upon.
4. *Focus on cultural issues*: A measurement program will affect the organization's culture; expect it and plan for it.

5. *Create a safe environment to collect and report true data*: Remember that without a good rationale people will be suspicious of new metrics, fearful of a time-and-motion study in sheep's clothing.
6. *Cultivate a predisposition to change*: The metrics will reveal deficiencies and inefficiencies so be ready to make improvements.
7. *Develop a complementary suite of measures*: Responding to an individual metric in isolation can have negative side effects. A suite of metrics lowers this risk.

If you believe that you are being metric mismanaged, you can try to instigate the above process by questioning management about why the metrics are being collected, how they are being used, and whether there is any justification for such use. You can also offer to provide corrective understanding of the metrics with opportunities of alternate metrics and appropriate use or more appropriate uses of the existing metrics.

Mushroom Management

Mushroom management is a situation in which management fails to communicate effectively with staff. Essentially, information is deliberately withheld in order to keep everyone "fat, dumb, and happy." The name is derived from the fact that mushrooms thrive in darkness and dim light but will die in the sunshine. As the old saying goes, "Keep them in the dark, feed them dung, watch them grow ... and then cut off their heads when you are done with them."

The dysfunction occurs when members of the team don't really understand the big picture; the effects can be significant, particularly with respect to requirements engineering when stakeholders get left out. It is somewhat insulting to assume that someone working on the front lines doesn't have a need to understand the bigger picture. Moreover, those who are working directly with customers, for example, might have excellent ideas that may have sweeping impact on the company. So, mushroom management can lead to low employee morale, turnover, missed opportunities, and general failure.

Those eager to perpetuate mushroom management will find excuses for not revealing information, strategy, and data. To refactor this situation some simple strategies to employ include:

- Take ownership of problems that allow you to demand more transparency.
- Seek out information on your own. It's out there. You just have to work harder to find it and you may have to put together the pieces. Between you and the other mushrooms, you might be able to see most of the larger picture.
- Advocate for conversion to a culture of open-book management.

With all refactoring, courage and patience are needed to affect change.

Other Paradigms for Requirements Management

Requirements Management and Improvisational Comedy

Improvisational comedy can provide some techniques for collaboration (and after all, requirements engineering is the ultimate in collaboration) and for dealing with adversity in requirements management. Anyone who has ever enjoyed improvisational comedy (e.g., the television show *Whose Line Is It Anyway?*) has seen the exquisite interplay of persons with very different points of view and the resolution of those differences. The author has studied improvisational comedy and has observed a number of lessons that can be taken away from that art.

- Listening skills are really important, both to hear what customers and other stakeholders are saying and to play off your partner(s) in the requirements engineering effort.
- When there is disagreement or partial agreement the best response is, "Yes, and ..." rather than, "Yes, but ..." That is, build on, rather than tear down ideas.
- Things will go wrong; both improvisational comedy and requirements engineering are about adapting.
- You should have fun in the face of adversity.
- Finally, you should learn to react by controlling only that which is controllable (usually, it is only your own reaction to certain events, not the events themselves).

You can actually practice some techniques from improvisational comedy to help you develop your listening skills, emotional intelligence, and ability to think on your feet, which in turn, will improve your practice as a requirements engineer and as a team player.

For example, consider one improvisational skill-building game called "Zip, Zap, Zup (or Zot)."* Here is how it works. Organize four or more people (the more, the better) to stand in a circle facing inside. One person starts off by saying either zip, zap, or zup. If that person looks at you, you look at someone else in the circle and reply in turn, zip, zap, or zup. Participants are allowed no other verbal communication than the three words. The "game" continues until all participants begin to anticipate which of the three responses is going to be given. The game is a lot harder to play than it seems, and the ability to anticipate responses can take several minutes (if it is attained at all). The point of this game/exercise is that it forces participants to "hear" emotions and pick up on other nonverbal cues.

* The author admits that this is also a traditional "drinking" game.

In another game, "Dr. Know-it-all," a group of three or more participants answers questions together, with each participant providing just one word of the answer at a time. So, in a requirements engineering exercise, we would gather a collection of participants from stakeholder group A and ask them to do the following: "Please complete the following sentence: the system should provide the following …" Then participants provide the answer one word each, in turn. This is a very difficult experience, and it is not intended as a requirements elicitation technique. It is a thought-building and team-building exercise and it can help to inform the requirements engineer and the participants about the challenges ahead.

One final exercise involves the requirements engineer answering questions from two other people at the same time. This experience helps the requirements engineer to think on her feet and also simulates what she will often experience when interacting with customers where she may need to respond simultaneously to questions from two different stakeholders/customers.

Although this author prefers comedy as the appropriate genre for requirements elicitation, others (e.g., Mayhaux and Maiden 2008) have studied improvisational theater as a paradigm for requirements discovery. In any case, it seems clear that our brains tend to suppress our best ideas and improvisation helps you to think spontaneously and generously. So, try these exercises for fun and to develop these important skills.

Requirements Management as Scriptwriting

In many ways the writing of screenplays for movies (scripts) has some similarities to requirements engineering. For example, Norden (2007) describes how requirements engineers can learn how to resolve different viewpoints by observing the screenwriting process. There are other similarities between the making of movies and software and some hardware/software systems. These include the following:

- Movies are announced well in advance while building certain expectations, which may not be delivered upon as the movie evolves (e.g., changes in actors, screenwriters, directors, plots, and release date). Software and systems are often announced in advance of their actual release and with subsequent changes in announced functionality and release date.
- Egos are often a significant factor in movies and software and systems.
- There are often too many hands involved in making movies and software and systems.
- Sometimes the needs of movies exceed the technology (and new technology has to be developed as a result). The same is true for software and systems.
- Movies often exceed budget and delivery expectations. Need we say more about software?
- Movies are filmed out of order, requirements are given, and software and some systems may be built this way, too.

- Movies are shot out of sequence and then assembled to make sense later. Software is almost always developed this way, too. Some systems are, too.
- A great deal of work ends up getting thrown away. In movies it ends up as film left on the cutting room floor, in software, as throwaway prototypes and one-time-use tools. Although systems components are not usually built to be thrown away, many test fixtures are.

What can we learn from big picture production? In a short vignette within Norden's article, Sara Jones provides the following tips for requirements engineers from screenwriting:

- Preparation is everything. Don't leap straight into the detail, but do your homework first!
- Expect to work through many drafts. Remember that different versions might each need to support planning and enable collaboration between stakeholders.
- Think in detail about the future system users—their backgrounds, attitudes, habits, likes, and dislikes.
- Don't just think about the users' physical actions. Remember what they might think (their cognitive actions) and feel.
- Remember your requirements story's arc. Requirements should tell the complete story of achieving user goals.
- Hold your audience—remember that someone must read your requirements document. Perhaps there is a place for tension and dramatic irony. (Norden, 2007)

Standards for Requirements Management*

There are many different international standards that provide either reference process models, management philosophies, or quality standards that can be used to inform the requirements engineering effort. These standards are not mutually exclusive in that they may be used in a complementary manner. Figure 9.2 summarizes the relevant software engineering-related standards and how they are related. In the figure ISO standards are shaded and relevant standards are grouped into the following categories:

- Improvement standards (metamodels) such as ISO 15504 (capability maturity determination and process assessment standard) and ISO 9001 (quality management)

* This section is excerpted from P.A. Laplante, *What Every Engineer Needs to Know About Software Engineering*, Boca Raton, FL: CRC/Taylor & Francis (2006) and C. Ebert, Requirements engineering: Management, In *Encyclopedia of Software Engineering*. P. Laplante (Ed.). Boca Raton, FL: Taylor & Francis, Published online: 932–948 (2010). With permission.

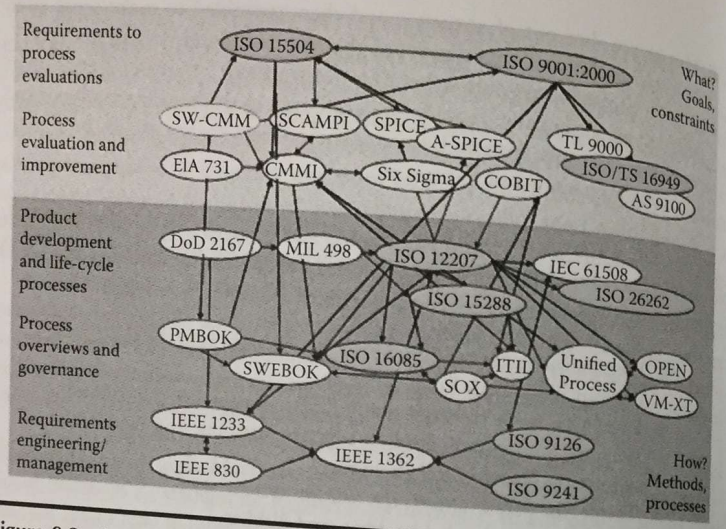

Figure 9.2 Standards applicable to requirements engineering. (Reprinted from C. Ebert, Requirements engineering: Management. In *Encyclopedia of Software Engineering*. P. Laplante (Ed.). Boca Raton, FL: Taylor & Francis, Published online: 932–948 (2010). With permission.)

- Quality management and improvement frameworks, such as CMMI (Capability Maturity Model Integrative) with SCAMPI (Standard CMMI Appraisal Method for Process Improvement), SPICE (Software Process Improvement and Capability Determination), ISO 15504, COBIT (Control Objectives for Information and related Technology) for IT organizations, Six Sigma, TL 9000, which details requirements and measurements to telecommunication quality management systems, ISO/TS (technical standard) 16949, which provides requirements to quality management in automotive systems, or AS (aerospace standard) 9100, which enhances ISO 9001 with specific requirements for the aerospace industry
- Process and life-cycle models, including DoD (US Department of Defense) 2167, MIL (military standard) 498, ISO 15288 (system life cycle), and ISO 12207 (software life cycle)
- Process implementation and governance regulations and policies such as Sarbanes–Oxley Act (SOX), standards such as ISO 16085 (risk management), de facto standards such as SWEBOK and PMBOK (Project Management Body of Knowledge), and various frameworks (Ebert 2010)

We have already covered SWEBOK in Chapter 1 and IEEE Std 830 in Chapters 4 and 5. We explore some of the other standards in the following sections.

ISO 9001

ISO Standard 9000 (International Standards Organization) is a generic worldwide standard for quality improvement. The standard, which collectively is described in five standards, ISO 9000 through ISO 9004, was designed to be applied in a wide variety of manufacturing environments. ISO 9001 through ISO 9004 apply to enterprises according to the scope of their activities. ISO 9004 and ISO 900X family are documents that provide guidelines for specific applications domains. These ISO standards are process-oriented, "common sense" practices that help companies create a quality environment.

ISO 9001, Quality Management Systems—Requirement, provides some guidance on requirements management within the discussion for "product realization." The standard calls for the following to be determined:

(a) Quality objectives and requirements for the product
(b) The need to establish processes, documents, and provide resources specific to the product
(c) Required verification, validation, monitoring, inspection, and test activities specific to the product and the criteria for product acceptance
(d) Records needed to provide evidence that the realization processes and resulting product meet requirements (see 4.2.4; ISO 9001 2008).

ISO 9000-3 does not provide specific process guidance for requirements management. The aforementioned recommendations, however, are helpful as a "checklist," especially when used in conjunction with appropriate requirements metrics (some of these are discussed in Chapter 8). In order to achieve certification under the ISO standard, significant documentation is required.

Six Sigma

Developed by Motorola, Six Sigma is a management philosophy based on removing process variation. Six Sigma focuses on the control of a process to ensure that outputs are within six standard deviations (six sigma) from the mean of the specified goals. Six Sigma is implemented using define, measure, analyze, improve, and control (DMAIC). An overview of a Six Sigma model for requirements management is depicted in Figure 9.3.

In Six Sigma *define* means to describe the process to be improved, usually through using some sort of business process model. In the case of requirements engineering we can think of this as identifying the required features that need improvement. *Measure* means to identify and capture relevant metrics for each aspect of the process model, in our case, the identified requirements. The goal-question-metric paradigm is helpful in this regard. *Analyze* means to study the captured metrics for opportunities for improvement. *Improve* means to change some aspect of the process so that beneficial changes are seen in the

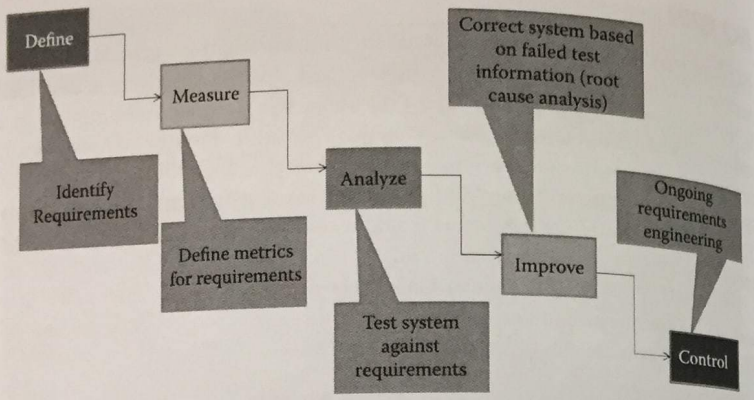

Figure 9.3 Six Sigma process for requirements engineering.

associated metrics, usually by attacking the aspect that will have the highest payback. Finally, *control* means to use ongoing monitoring of the metrics to continuously revisit the model, observe the metrics, and refine the process as needed.

Six Sigma is more process yield-based than the CMMI so the CMMI process areas can be used to support DMAIC (e.g., by encouraging measurement). And, although CMMI identifies activities, Six Sigma helps optimize those activities. Six Sigma can also provide specific tools for implementing CMMI practices (e.g., estimation and risk management).

Some organizations use Six Sigma as part of their software quality practice. The issue here, however, is in finding an appropriate business process model for the software production process that does not devolve into a simple, and highly artificial, waterfall process. If appropriate metrics for requirements can be determined (e.g., 830) then Six Sigma can be used to improve the RE process.

Capability Maturity Model Integrative (CMMI)

The Capability Maturity Model Integrative is a systems and software quality model consisting of five levels. The CMMI is not a life-cycle model, but rather a system for describing the principles and practices underlying process maturity. CMMI is intended to help organizations improve the maturity of their processes in terms of an evolutionary path from ad hoc chaotic processes to mature disciplined processes.

Developed by the Software Engineering Institute at Carnegie Mellon University, the CMMI is organized into five maturity levels. Predictability, effectiveness, and control of an organization's software processes are believed to improve as the

organization moves up these five levels. Although not truly rigorous, there is some empirical evidence that supports this position.

The Capability Maturity Model describes both high-level and low-level requirements management processes. The high-level processes apply to managers and team leaders, and the low-level processes pertain to analysts, designers, developers, and testers.

Typical high-level requirements practices/processes include:

- Adhere to organizational policies.
- Track documented project plans.
- Allocate adequate resources.
- Assign responsibility and authority.
- Train appropriate personnel.
- Place all items under version or configuration control and have them reviewed by all (available) stakeholders.
- Comply with relevant standards.
- Review status with higher management.

And low-level best practices/processes include:

- Understand requirements.
- Get all participants to commit to requirements.
- Manage requirements changes throughout the life cycle.
- Manage requirements traceability (forward and backward).
- Identify and correct inconsistencies between project plans and requirements.

These practices are consistent with those that have been discussed throughout the text.

Achieving level 3 and higher for the CMMI requires that these best practices be documented and followed within an organization.

The CMM family of quality models defines requirements management as:

> Establishing and maintaining an agreement with the customer on the requirements for the software project. The agreement forms the basis for estimating, planning, performing, and tracking the software project's activities throughout the software life cycle. (CMM 1993)

IEEE 830

We have already discussed IEEE Standard 830 in the context of risk mitigation. In terms of managing the requirements engineering activities, standard 830 provides two major things. First, the standard describes the qualities that govern good software requirements specifications. In addition, 830 provides a framework for approaching the organization of requirements (e.g., object-oriented, hierarchical, etc.).

From a requirements management perspective, the "sample table of contents" is the least important offering of 830.

ISO/IEEE 12207

ISO 12207: Standard for Information Technology—Software Life Cycle Processes, describes five "primary processes": acquisition, supply, development, maintenance, and operation. ISO 12207 divides the five processes into "activities," and the activities into "tasks," while placing requirements upon their execution. It also specifies eight "supporting processes"—documentation, configuration management, quality assurance, verification, validation, joint review, audit, and problem resolution—as well as four "organizational processes" of management, infrastructure, improvement, and training.

The ISO standard intends for organizations to tailor these processes to fit the scope of their particular projects by deleting all inapplicable activities, and it defines 12207 compliance as being in terms of tailored performance of those processes, activities, and tasks.

12207 provides a structure of processes using mutually accepted terminology, rather than dictating a particular life-cycle model or software development method. Inasmuch as it is a relatively high-level document, 12207 does not specify the details of how to perform the activities and tasks comprising the processes. Nor does it prescribe the name, format, or content of documentation. Therefore, organizations seeking to apply 12207 need to use additional standards or procedures that specify those details.

The IEEE recognizes this standard with the equivalent numbering: "IEEE/EIA 12207.0-1996, IEEE/EIA Standard Industry Implementation of International Standard ISO/IEC12207:1995, and (ISO/IEC 12207) Standard for Information Technology—Software Life Cycle Processes." ISO/IEC 15504 Information Technology—Process Assessment, also known as SPICE, is a derivative of 12207.

Usage of Standards

The Marinelli (2008) survey provided some data on the use of reference standards as part of their software quality management (SQM) approaches in organizations. In that survey, 59% of respondents either did not use reference standards or were not aware of the applicable standards applied. The relative utilization of the different reference standards used is shown in Figure 9.4.

Of the 41% of respondents who reported using one of the SQM approaches available, ISO9001, Six Sigma, and CMM/CMMI were used in 22.22% of projects. ISO 15504/SPICE and GAMP (Good Automated Manufacturing Practice) is a quality framework used in the automotive and pharmaceutical industries.

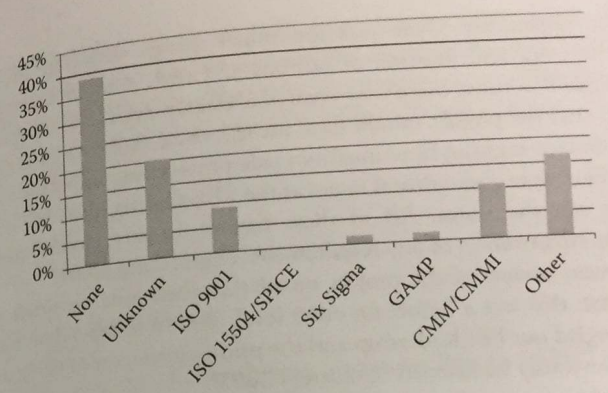

Figure 9.4 Relative prevalence of software quality management reference standards used in industry. (Reprinted from V. Marinelli, *An Analysis of Current Trends in Requirements Engineering Practice*, Master of Software Engineering Professional Paper, Penn State University, Malvern (2008). With permission.)

A Case Study: FBI Virtual Case File

We close this chapter with a brief case study involving the US Federal Bureau of Investigation's (FBI) disastrous Virtual Case File system, which was a spectacular failure of requirements management, among other things. This discussion is largely based on Goldstein's report (2005), Alfonsi's analysis (2005), and Israel's project retrospective (2012).

Begun in 2000, the Virtual Case File (VCF) was supposed to automate the FBI's paper-based work environment and allow agents and intelligence analysts to share investigative information. The system would also replace the obsolete Automated Case Support (ACS), which was based on 1970s software technology. After 12 years and $451 million the project is not complete and may never be completed. The VCF may eventually become fully operational, but by all accounts, the project has been a financial and political failure.

What went wrong with VCF? Quite a bit, some of it political, some of it due to the complexities of a mammoth bureaucracy, but some of the failures can be attributed to bad requirements engineering. Goldstein noted that the project was poorly defined from the beginning and that it suffered from slowly evolving requirements (an 800-page document to start). And even though JAD sessions were used for requirements discovery (several two-week sessions over six months), these were used late in the project and only after a change of vendors. Apparently, there was no discipline to stop requirements from snowballing. And an ever-changing set of sponsors (changes in leadership at the FBI as well as turnover in key congressional supporters) caused requirements drift.

Moreover, apparently, there was too much design detail in SRS, literally describing the color and position of buttons and text. And many requirements failed the simple 830 rules: often they were not "clear, precise, and complete." Israel (2012) noted that the project should have incorporated extensive prototyping, but pressures to deliver working functionality made prototyping impossible.

Finally, there were also cultural issues at the FBI; sharing ideas and challenging decisions are not the norm. All of these factors, in any environment, would jeopardize the effectiveness of any requirements engineering effort.

A subsequent review of the project noted that there was no single person or group to blame: this was a failure on every level. But an independent fact-finding committee singled out FBI leadership and the prime contractor as being at fault for the reasons previously mentioned (Goldstein 2005).

The project was a complex one and according to former FBI CIO Israel, beyond the capabilities of the organization to manage (Israel 2012). The project went through multiple changes of CIOs and project leaders. Originally begun as an internal project, the FBI realized that it did not have the necessary expertise to complete it, and in 2007, handed the project to Lockheed Martin. But by 2010, after many setbacks, the FBI reclaimed the project and at this writing is expecting to complete the project using an agile approach (Israel 2012).

Aside from the political ramifications, and the usual problems when designing large complex systems, what can be learned from the failure of VCF? From a requirements engineering standpoint, there are three very specific lessons. First, don't rush to get a system out before elicitation is completed. Obviously, there are pressures to deliver, particularly for such an expensive and high-profile system, but these pressures need to be resisted by strong leadership. Second, the system needs to be defined "completely and correctly" from the beginning. This was highly problematic for the VCF due to the various changes in FBI leadership during the course of its evolution, however, a well-disciplined process could have helped to provide pushback against late requirements changes. Finally, for very large systems it is helpful to have an architecture in mind. We realize that it has been said that the architecture should not be explicitly incorporated in the requirements specification, but having an architectural target can help inform the elicitation process, even if the imagined architecture is not the final one.

Exercises

9.1 Should a request to add or change features be anonymous?
9.2 How could metrics abuse begin to develop in an organization?
9.3 Give an example of process clash, from your own experience, if possible.
9.4 Give an example of metrics abuse, from your own experience, if possible.

9.5 Give an example of divergent goals, from your own experience, if possible.
9.6 How can CMMI be used to identify and reconcile process clash?
9.7 With a group of friends, classmates, or teammates, use Wideband Delphi to select a restaurant to share a meal from the following choices:
- Olive Garden
- Carrabba's
- Red Lobster
- Golden Corral
- PF Chang's
- Ruth's Chris Steak House
- Cracker Barrel

9.8 Investigate and use the Analytical Hierarchy Process (AHP) to resolve the decision-making problem in Exercise 9.7.

References

Alfonsi, B. (2005). FBI's virtual case file living in limbo, *Security & Privacy*, 3(2): 26–31.

Andriole, S. (1998). The politics of requirements management, *IEEE Software*, November/December, pp. 82–84.

Bhat, J.M., Gupta, M., and Murthy, S.N. (2006). Overcoming requirements engineering challenges: Lessons from offshore outsourcing, *IEEE Software*, September/October, pp. 38–44.

Brown, W.J., Malveau, R.C., McCormick, H.W., and Mowbray, T.J. (1998). *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, New York: John Wiley and Sons.

Dekkers, C.A. and McQuaid, P.A. (2002). The dangers of using software metrics to (mis) manage, *IT Professional*, 4(2): 24–30.

Ebert, C. (2010). Requirements engineering: Management. In *Encyclopedia of Software Engineering*, P. Laplante (Ed.). Boca Raton, FL: Taylor & Francis. Published online: 932–948.

Goldstein, H. (2005). Who killed the virtual case file? *Spectrum*, 42(9): 24–35.

Hull, E., Jackson, K., and Dick, J. (2011). Management aspects of requirements engineering. In *Requirements Engineering*, New York: Springer, pp. 159–180.

ISO 9001 Quality Management Systems—Requirements (2008). http://www.iso.org, last accessed March 2013.

Israel, J.W. (2012). Why the FBI can't build a case management system, *Computer*, 45(6): 73–80.

Laplante, P.A. (2006). *What Every Engineer Needs to Know about Software Engineering*, Boca Raton, FL: CRC/Taylor & Francis, 2006.

Lawrence, B. (1996). Unresolved ambiguity, *American Programmer*, 9(5): 17–22.

Marinelli, V. (2008). *An Analysis of Current Trends in Requirements Engineering Practice*, Master of Software Engineering Professional Paper, Penn State University, Great Valley School of Graduate Professional Studies, Malvern, PA.

Mayhaux, M. and Maiden, N. (2008). Theater improvisers know the requirements game, *Software*, September/October, pp. 68–69.

Neill, C.J., Laplante, P.A., and DeFranco, J. (2012). *Antipatterns: Identification, Refactoring, and Management*, second edition. Boca Raton, FL: CRC Press.

Norden, B. (2007). Screenwriting for requirements engineers, *Software*, July, pp. 26–27.

Paulk, M., Weber, C.V., Garcia, S.M., Chrissis, M-B.C., and Bush, M. (1993). CMM Practices Manual, CMU/SEI-93-TR-25, L2, http://repository.cmu.edu/cgi/viewcontent.cgi?article = 1179&context = sei, last accessed March 2013.

Reinhart, G. and Meis, J.F. (2012). Requirements management as a success factor for simultaneous engineering. *Enabling Manufacturing Competitiveness and Economic Sustainability*: 221–226.

Sinha, V. and Sengupta, B. (2006). Enabling collaboration in distributed requirements management, *Software*, September/October: 23 (5): 52–61.