Chapter 8

Tool Support for Requirements Engineering

Introduction

We have already discussed automated requirements analysis via the NASA ARM tool in Chapter 5. But word processors, databases, spreadsheets, content analyzers, concept mapping programs, automated requirements checkers, and so on are all tools of interest to the requirements engineer. But the most celebrated requirements tools are large commercial packages that provide a high level of integrated functionality. Table 8.1 compares the capabilities of conventional office tools used for requirements management to a feature-rich requirements management tool.

The chief feature of requirements management tools is the ability to represent and organize all "typical" requirements engineering objects such as use cases, scenarios, and user stories. Support for user-defined entities is also usually provided. Other typical functionalities for these large, commercial requirements engineering tools include the following:

- Multiuser support and version control
- Online collaboration support
- Customizable user interfaces
- Built-in support for standards templates (such as IEEE 12207 and IEEE 830)
- Verification and validation tools
- Customizable functionality through a programmable interface
- Support for traceability
- User-defined glossary support (Heindl et al. 2006)

Table 8.1 Requirements Repository Metric Capabilities

	Word Processor	*Spreadsheet*	*Relational Database*	*Requirements Management Tool*
Document Size	Yes	No	No	Not in preformatted state
Dynamic Changes Over Time	Possible with complex change tracking enabled	No	No	Yes
Release Size	Yes	Yes	Yes	Yes
Requirement Expansion Profile	No	No	Yes	Yes
Requirements Verification	No	No	Possible	Possible
Requirements Volatility	Yes	Yes	Yes	Yes
Test Coverage	No	Possible with complex equation logic	Yes	Yes
Test Span		Possible with complex equation logic	Yes	Yes
Test Types	Yes	Yes	Yes	Yes

Source: Adapted from Hammer et al., *CROSSTALK The Journal of Defense Software Engineering*, 20–25, 1998.

Verification and validation features are an important component of any automated requirements engineering tool. Indeed, the more sophisticated commercial requirements engineering tools provide other requirements checking, tracing, and archiving features. These are shown in Table 8.2.

These features are particularly important because they provide for accurate tracing of artifacts over time. Traceability is an important characteristic of the SRS document and bears further discussion.

Table 8.2 Automated Requirements Engineering Tool Features

Tool Feature	*Description*
Definition of workflow for requirements	A workflow (states, roles, state transitions) is configurable for requirements.
Automated generation of bidirectionality of traces	When the user creates a trace between artifact A and artifact B it automatically establishes a backward trace from B to A.
Definition of user-specific trace types	An authorized user can define trace types and assign names.
Suspect traces	When a requirement changes, the tool automatically highlights all traces related to this requirement for checking and updating traces.
Long-term archiving functionality	All data in the tool can be archived in a format accessible without the tool if necessary.

Source: Heindl et al. (2006).

Traceability Support

Traceability is concerned with the relationships among requirements, their sources, and numerous other artifacts. We define three kinds of traceability: source traceability, which links requirements to stakeholders who proposed these requirements; requirements traceability, which links between dependent requirements; and design traceability, which provides links from the requirements to the design.

Within the SRS document, traceability focuses on the interrelationship between requirements and their sources, stakeholders, standards, regulations, and so on. For these purposes traceable artifacts include:

- Requirement
- Stakeholder(s) who are associated with a requirement
- Standard, regulation, or law that mandates the requirement
- Rationale (for the requirement)
- Keywords (for searching)

The various combinations of these and other artifacts lead to many traceability matrix formats, a few of which are presented shortly. Yue, Briand, and Labiche (2011) also suggest creating traceability matrices that maintain links between requirements elements and analysis model elements. An extensive discussion of requirements tracing approaches and applications can be found in Lee, Gandhi, and Park (2011).

Requirements Linkage Traceability Matrix

It is not unusual for one requirement to have an explicit reference to another requirement either as a "uses" or "refers to" relationship. Consider the following requirement:

> 2.1.1 The system shall provide for control of the heating system and be schedulable according to the time scheduling function (ref. Requirement 3.2.2).

which depicts a "refers to" relationship. The "uses" relationship is illustrated in the following requirement:

> 2.1.1 The system shall provide for control of the heating system in accordance with the timetable shown in Requirement 3.2.2 and the safety features described in Requirement 4.1.

The main distinction between "uses" and "refers to" is that the "uses" represents a stronger direct link.

The primary artifact of traceability is the requirements traceability matrix. The requirements traceability matrix can appear in tabular form in the SRS document, in a standalone traceability document, or internally represented within a tool for modification, display, and printout. There are many variations of traceability matrices depending on which artifacts are to be included.

One form of matrix shows the interrelationship between requirements. Here the entries in the requirements traceability matrix are defined as follows:

$R_{ij} = R$ if requirement i references requirement j (meaning "refers to" for informational purposes).

$R_{ij} = U$ if requirement i uses requirement j (meaning "depends on" directly).

R_{ij} = blank otherwise.

When a requirement both uses and references another requirement, the entry "U" supersedes that of "R." Because self-references are not included, the diagonal of the matrix always contains blank entries. We would also not expect circular referencing in the requirements so that if $R_{ij} = U$ or $R_{ij} = R$, we would expect R_{ji} to be blank. In fact, an automated verification feature in a requirements engineering tool should flag such circular references.

The format of a typical requirements traceability matrix is shown in Table 8.3.

A partial traceability matrix for a hypothetical system with sample entries is shown in Table 8.4.

Table 8.3 Format of Traceability Matrix (R) for Requirements Specification

Requirement ID	1	1.1	1.1.1	...	1.2	1.2.1	...	2	2.1	2.1.1	...
1											
1.1											
1.1.1											
...											
1.2											
1.2.1											
....											
2											
2.1											
2.1.1											
...											

Inasmuch as R is usually sparse, it is convenient to list only those rows and columns that are nonblank. For example, for the SRS document for the smart home found in Appendix A, the traceability matrix explicitly derived from the requirements is shown in Table 8.5.

A sparse traceability matrix of this type indicates a low level of explicit coupling between requirements. A low level of coupling within the requirements document is desirable: the more linked requirements, the more changes to one requirement propagate to others. In fact, explicit requirements linkages are a violation of the principle of separation of concerns, a fundamental tenet of software engineering. Therefore, link requirements only when absolutely necessary.

Other kinds of linkages are desirable, however. As noted in Chapter 5, it is expected that each requirement will be linked to multiple tests. Test span metrics are used to characterize the linkages between the SRS document and the test plan in order to identify insufficient or excessive testing. The typical test span metrics are:

- Requirements per test
- Tests per requirement

And there is a tradeoff between time and cost of testing versus the comprehensiveness of testing. Planning for testing is part of the requirements engineering effort,

Table 8.4 Partial Traceability Matrix for a Fictitious System Specification

Requirement ID	1	1.1	1.1.1	1.1.2	1.2	1.2.1	1.2.2	2	2.1	2.1.1	3
1									R	R	
1.1			U								
1.1.1	R							R			
1.1.2						R					
1.2										U	
1.2.1							R				
1.2.2										R	
2	U								R		
2.1				U							
2.1.1											R
3					R						

Table 8.5 Traceability Matrix for Smart Home Requirements Specification Shown in Appendix A

Requirement ID	5.11	9.11
9.1.1	R	
9.10.7		R

as we have discussed exhaustively, and these linkages need to be anticipated in the SRS document.

Requirements Source Traceability Matrix

Another kind of traceability matrix links requirements to their sources. Aside from those coming directly from users, many requirements are derived from governmental regulations and from standards. Linking the requirements to these sources can be very helpful when these sources change. Table 8.6 shows the typical format for such a traceability matrix.

This kind of traceability matrix is especially useful for tracking nonfunctional requirements.

Table 8.6 Traceability Matrix Showing Requirements and Their Sources

Requirement ID	Federal Regulation #1	Federal Regulation #2	State Regulation #1	State Regulation #1	International Standard #1
3.1.1.3	X	–	–	–	–
3.1.2.9	X	X	–	–	–
3.2.1.8	–	–	X	X	–
3.2.2.5	–	–	X	–	–
3.2.2.6	–	–	–	–	X
3.3.1	–	X	–	–	–
3.3.2	–	X	–	–	–
3.4.1	–	X	–	–	–
3.4.3	–	X	–	–	–
3.4.4	–	X	–	–	–
3.6.5.1	–	–	–	X	–
3.6.6.4	–	–	–	–	X

Table 8.7 Traceability Matrix Showing Requirements, Their Rank, and Their Stakeholder Source

Requirement	Rank 1 (Lowest Importance) – 5 (Highest Importance)	Stakeholder Source D – Doctor, N – Nurse, A – Administrative Support Staff, P – Patient, R – Regulator
3.1.1.1	5	R
3.1.1.2	4	R
3.1.2.1	5	D,N,A
3.1.2.2	3	D,N,A

Requirements Stakeholder Traceability Matrix

Another kind of traceability matrix links stakeholders to the requirements they submitted, an example of which is shown in Table 8.7. Table 8.7 also incorporates a ranking schema, which greatly facilitates requirements negotiation and tradeoff analysis.

Only three kinds of traceability matrices have been shown in the previous sections but any variation or combination of these can be created and they can incorporate the other traceability artifacts mentioned. Commercial requirements engineering tools allow the user to customize all kinds of traceability matrices.

Requirements Management Tools

Requirements tools are extremely important in managing requirements information across the life cycle of a project. Although it is not the intent to promote any particular commercial requirements engineering tool, it is appropriate to mention one example that embodies most of the features that are found in many others. IBM's DOORS (dynamic object-oriented requirements system) is one of the most widely used requirements management tools. DOORS was developed by Rational Software and the company and its products were acquired by IBM in 2003. DOORS is essentially an object database for requirements management. That is, each feature is represented as an object containing a feature description, feature graph, and use case diagram. The database is further organized by creating folders and projects and a history of all module and object-level actions is maintained. There is an extension called Analyst, which integrates DOORS with another tool that allows UML models to be embedded and traced within DOORS (Hull, Jackson, and Dick 2011).

Feature attributes include mandatory, optional, single adaptor, and multiple adaptor, and the user can specify whether an attribute is required or excluded, which embodies the complementary and supporting nature of requirements. Other attributes are based on use case packages and product instances (Eriksson et al. 2005).

DOORS offers linking between all objects in a project for full traceability and missing link analysis. The tool can be customized via a C-like programming language.

Standard templates are available to structure requirements in compliance with the ISO 12207, ISO 6592, and IEEE 830 software standards (Volere 2013).

There are many commercial and open source tools, and it is important to evaluate the features of any tool carefully before adopting it in the enterprise. Sud and Arthur (2003) evaluated a number of current requirements management tools using the following dimensions:

- Requirements traceability mechanism
- Requirements analysis mechanism
- Security and accessibility mechanism
- Portability and backend compatibility
- Configuration management approach
- Communication and collaboration mechanism
- Change management support

- Online publishing support
- Usability features such as word processor compatibility
- SRS documentation format

These dimensions can be used to compare various commercial and open source requirements management tools.

Open Source Requirements Engineering Tools

As of this writing, there are more than 324,000 open source projects including all kinds of tools, utilities, libraries, and games. Not surprisingly, then, there are many open source solutions for requirements engineering, some as good as or almost as good as their commercial equivalents. We take a look at three open source utilities that can be of use to the requirements engineer. This selection is a tiny sampling; you are encouraged to turn first to open source repositories to look for tools before purchasing or trying to develop them from scratch.

FreeMind

Mind-mapping tools allow the user to create concept maps, which depict the relationship between things and ideas and are useful for the requirements engineer for brainstorming, task analysis, laddering, traceability, JAD, focus groups, and many other requirements engineering activities. Concept-mapping tools allow for the easy creation of concept maps and although the foregoing discussion is based on a specific tool for illustrative purposes, any concept-mapping tool can be used by the requirements engineer.

FreeMind is an open source mind-mapping tool written in Java (FreeMind 2013). The tool has an easy-to-use graphical user interface (GUI). Concepts are mapped in a hierarchical structure based on a primary concept and subordinate ideas to that concept. Handy icons can be used to mark priorities, important criteria, or hazards for each idea. Many other icons are available to enrich the content of the mind map.

To illustrate the use of the requirements engineer's use of mind maps, let's look at Phil's concept of the smart home control system. Starting off with the basic concept of a "smart home" Phil creates a parent node (represented by a bubble) as shown in Figure 8.1.

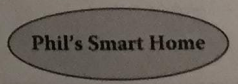

Figure 8.1 Phil's smart home system.

Next, Phil uses the tool to attach a feature to the system (security) and assign a priority to that feature. Although there are up to seven levels of rankings available, Phil chooses a simple three-level system:

1. Mandatory
2. Optional
3. Nice to have

The resultant ranked feature and updated mind map is shown in Figure 8.2.

Phil then adds details to the feature. For example, he wants the security system of the smart home to use and work with the existing security system in the home. Another feature he desires is for the HVAC (heating, ventilation, and air-conditioning) system to integrate with the smart home system. In addition, because Phil has delicate plants, pets, and collectibles in the house, it is important that the temperature never exceed 100° Fahrenheit, so this hazard is marked with a bomb icon. The revised mind map is shown in Figure 8.3.

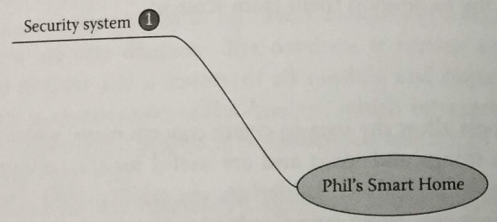

Figure 8.2 Adding the first feature to Phil's smart home system.

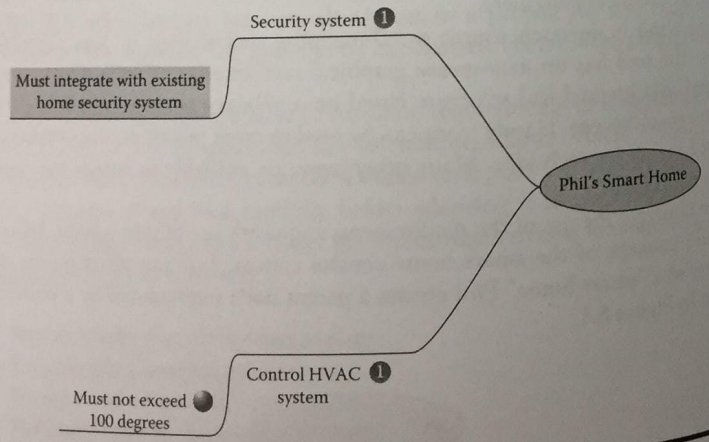

Figure 8.3 Adding a detail to the first feature of Phil's smart home system and adding a hazard.

Phil continues with his brainstorming of features, adding music management, lawn care, and a telephone answering system. Some of the features include important details that are marked with the appropriate symbol (Figure 8.4).

As he sees the system materialize in a visual sense, Phil thinks of more features and is able to make the appropriate prioritization decisions. Phil adds to his mind map accordingly, as seen in Figure 8.5.

The mind map shown in Figure 8.5 is not complete. But this mind map can be used during elicitation discussions with requirements engineers and can be easily evolved over time.

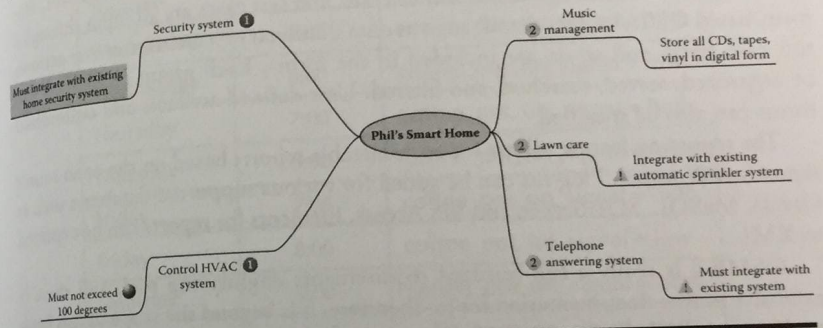

Figure 8.4 Adding more features to Phil's smart home system.

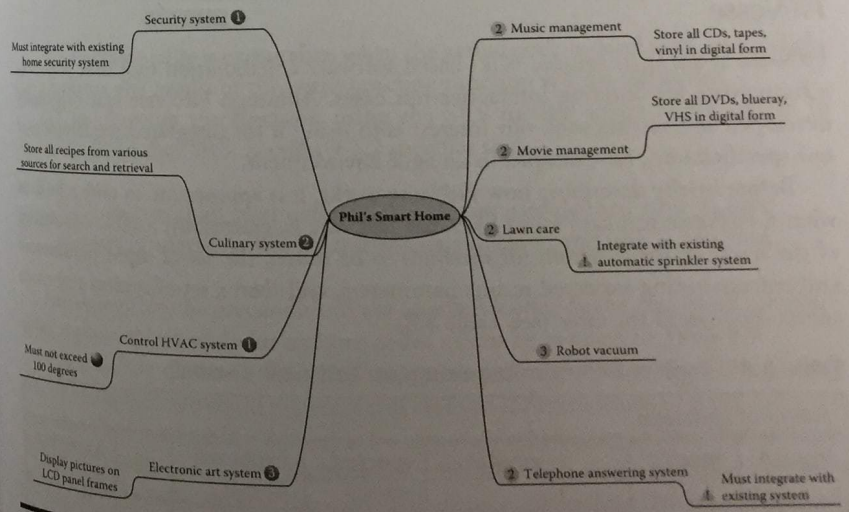

Figure 8.5 Phil's partial mind map of a smart home system.

Of course, there are other open source and commercial "mind- mapping" tools that have similar functionality to FreeMind. We are simply suggesting that such a tool can be very valuable for many requirements elicitation activities.

Open Source Requirements Management Tool (OSRMT)

The Open Source Requirements Management Tool (OSRMT) is a platform-independent program written in Java and uses Swing and JBoss to provide full life-cycle support for requirements development and traceability. In OSRMT, features, requirements, design items, source code, and test cases are all input through a form-based GUI. Artifact details such as data dictionary elements, use case actions, test case steps, and so on are included in the forms. Each group of artifacts can be organized, sorted, searched, and filtered. User-defined artifacts and customized forms can also be specified.

The reporting feature allows for customizable reports based on the open source report writer Jasper. Plug-ins can be added for various supported databases such as Oracle, MySQL, SQL Server, and MS Access. Elements for reports can be exported to XML.

OSMRT is really a full-featured requirements engineering tool, and there is extensive online documentation for it. Therefore, it is beyond the scope of this book to discuss installation and use of this open source tool. But the continued emergence of such free-for-use solutions may challenge commercial for-fee solutions and therefore cannot be ignored (OSRMT).

FitNesse

FitNesse is an open source, Wiki-based software collaboration tool that provides a framework for building interactive test cases. Although FitNesse was originally developed as a testing tool, our interest is in using it for integrated requirements/test specifications, for example, in an agile environment.

Before briefly describing how FitNesse works, it is appropriate to take a look at what a FitNesse test case looks like. The test case is, essentially, a table consisting of the name of the function (or method) to be tested, the set of input parameters and corresponding expected results parameters, and then a set of typical test cases across the rows of the table (see Table 8.8).

Table 8.8 Typical FitNesse Requirement/Test Case Format

Name of Function							
Input 1	Input 2	…	Input *n*	Expected Result 1	Expected Result 2	…	Expected Result *m*

Table 8.9 FitNesse Requirements Specification/Test Case for Activate Coffee Pot Function in Smart Home System

Activate_Coffee_Pot		
Day	*Time*	*Output*
Monday	7:00	coffee_pot_on_signal = high
Monday	8:00	coffee_pot_on_signal = low
Tuesday	7:00	coffee_pot_on_signal = high
Tuesday	8:00	coffee_pot_on_signal = low
Wednesday	7:00	coffee_pot_on_signal = high
Wednesday	8:00	coffee_pot_on_signal = low
Thursday	7:00	coffee_pot_on_signal = high
Thursday	8:00	coffee_pot_on_signal = low
Friday	7:00	coffee_pot_on_signal = high
Friday	8:00	coffee_pot_on_signal = low
Saturday	9:00	coffee_pot_on_signal = high
Saturday	10:00	coffee_pot_on_signal = low
Sunday	10:00	coffee_pot_on_signal = high
Sunday	11:00	coffee_pot_on_signal = low

The FitNesse table is executable; when the actual code is developed, upon clicking a button, the results of the corresponding function will be calculated.

Let us illustrate the situation using the SRS for the smart home system in Appendix A. In that SRS, Section 9.2.2 specifies that

> 9.2.2 System shall start coffee maker at any user-defined time as long as water is present, coffee bean levels are sufficient, and unit is powered.

Assuming that the preconditions are satisfied (coffee beans and water are available), a plausible schedule for making coffee might be as given in Table 8.9.

The schedule shown in Table 8.9 recognizes that the homeowner likes to sleep late on weekends. We could populate this table all different kinds of ways. Now it is true that the functionality desired is for dynamic scheduling so that, for example, during a holiday week, the homeowner can sleep late every day. But the table here is used to indicate one scenario of appliance scheduling. In fact, if properly configured via the FitNesse framework, Table 8.9 constitutes an executable test case for the finished system.

Without going into too much detail, here is roughly how FitNesse works. FitNesse is a Wiki GUI built on top of the Fit framework. Fit runs the fixture code (a Java or C# class) corresponding to the test table. For example, in the top row of Table 8.9, "Activate_Coffee_Pot" specifies the actual class to be called. FitNesse colors the expected results cells red, green, or yellow, if the test failed, passed, or an exception was thrown, respectively (Gandhi et al. 2005).

Requirements Engineering Tool Best Practices

Whatever requirements engineering tool(s) you use, it is appropriate to use the tool judiciously and follow certain best practices. One set of such best practices is offered by Cleland-Huang et al. (2007):

Trace for a purpose. That is, determine which linkages are truly important; otherwise, a large number of extraneous links will be generated.
Define a suitable trace granularity. For example, linkages should be placed at the appropriate package, class, or method level.
Support in-place traceability. Provide traceability between elements as they reside in their native environments.
Use a well-defined project glossary. Create the glossary during initial discovery meetings with stakeholders and use consistently throughout the requirements engineering process.
Write quality requirements. Make sure to follow generally accepted best practices such as IEEE 830, which are particularly important for traceability.
Construct a meaningful hierarchy. Experimental results show that hierarchically organized requirements are more susceptible to intelligent linking software.
Bridge the intra-domain semantic gap. For example, avoid overloaded terminology, that is, words that mean completely different things in two different contexts.
Create rich content. Incorporate rationales and domain knowledge in each requirement.
Finally, be sure to use a process improvement plan for improving the requirements engineering process.

Elicitation Support Technologies

We close this chapter by remarking on some technologies that can be used to support various requirements elicitation processes and techniques previously discussed. These technologies include:

- Wikis
- Mobile technologies
- Virtual environments
- Content analysis

Using Wikis for Requirements Elicitation

Wikis are a collaborative technology in which users can format and post text and images to a Web site. Access control is achieved through password protection and semaphore-like protection mechanisms (i.e., only one user can write to a particular page of the site at any one time).

Wikis can be used for collaboration, for example, to facilitate group work, card entry (for card sorting), template completions, surveys, and for organizing the requirements document. Moreover, Wiki-based requirements can be exported directly to publishing tools and validation tools. In addition, Wikis can be used to build interactive documents that can help to automate test cases (recall that requirements specifications should contain acceptance criteria for all requirements). For example, FitNesse is a free, Wiki-based software collaboration GUI built on top of the Fit framework. FitNesse provides a framework for building Web-based requirements documentation with embedded, interactive test cases (FitNesse 2007).

Mobile Technologies

Various mobile technologies such as cell phones and personal digital assistants can be used to capture requirements information in situ. For example, while physicians are working with patients, they can transmit information about the activities they are conducting directly to the requirements engineer, without the latter having to be on site. Using mobile devices is particularly useful because they enable instantaneous recording of ideas and discoveries. Such an approach can support brainstorming, scenario generation, surveys, and many other standard requirements elicitation techniques even when the customer is not easily accessible (such as in off-shore software development situations).

Lutz, Schäfer, and Diehl (2012) developed an application called CREWSpace that allowed users to role-play through Android-enabled devices such as smartphones and tablets and in doing so, to interact with a representation model displayed on a shared screen. By keeping track of the role-playing states, the software was able to create Class Responsibility Collaboration (CRC) cards, which are a brainstorming tool used in object-oriented software systems. The CRC cards could then be incorporated into the requirements document.

A good discussion of the emergence of the use of mobile technologies in requirements discovery can be found in Maiden et al. (2007).

Virtual Environments

Virtual world environments are complex simulations of real-world situations using enhanced graphics and various specialized devices such as tactile pressure sensors,

force feedback peripherals, and three-dimensional display glasses. Virtual world environments can be used to provide realism in testing, validating, and agreeing requirements particularly for novel or complex application environments. Russell and Creighton (2011) suggest that using virtual environments can do the following:

- Clarify current shortcomings.
- Draw attention to potential benefits.
- Show before-and-after conditions.
- Personalize the impressions of various actors and roles.
- Create a more shared appreciation of the effective components of valuation, exploration of alternative timelines.

For example, in the airline baggage handling system a three-dimensional layout of the relevant portions of the airport can be constructed and stakeholders can use characters (avatars) to traverse the digital scene and validate requirements that have been previously communicated.

Content Analysis

Content analysis is a technique used in the social sciences for structuring and finding meaning in unstructured information. That is, by analyzing writing or written artifacts of speech, things of importance to the stakeholder can be obtained. In analyzing these writings the objective is to identify recurrent themes. By written artifacts we mean transcripts from group meetings, unstructured interviews, survey data, or e-mails (any text or artifact that can be converted to text).

Content analysis can be done manually by tagging (colored highlighters) identical words and similar phrases that recur in various writings. For example, Figure 8.6 contains a sample content analysis of some text. The text is an excerpt

I am gathering requirements for **a smart home** for **a customer.**

I spend **long periods of time** interviewing **the customer** about what **she wants.**

I spend **time** interacting with **the customer** as **she** goes about **her** day and ask questions such as "Why are you running the dishwasher at night? Why not in the morning?"

I spend long **periods of time** passively observing **the customer** "in action" in her current home to get nonverbal clues about **her** wants and desires.

I gain other information from **the home** itself – the **books** on the **book** shelf, paintings on the wall, furniture styles, evidence of hobbies, signs of wear and tear on various appliances, etc.

Figure 8.6 Sample content analysis of some random text.

from a notebook of a requirements engineer who has interviewed a customer regarding a smart home.

As we read the text we begin noticing recurring themes, and as we do so, we highlight them with a different colored marker. For example, the word "I" is mentioned repeatedly, and for whatever reason, we decide that this noun hints at an important theme, and we highlight it in one color. We then notice that "smart home" or "the home" is mentioned several times; we highlight these with a different color from the previous theme. The notion of time is also mentioned frequently ("long periods of time," "time," and "periods of time") and so we highlight these with a consistent color. And so on.

Free and for-fee tools for content analysis can be used to automate this process.

Metrics

Metrics can be calculated or collected at various stages of the requirements engineering life cycle in order to monitor and improve the requirements engineering processes and practices. Costello and Liu (1995) describe the following types of requirements metrics:

- Volatility
- Traceability
- Completeness
- Defect density
- Fault density
- Interface consistency
- Problem report and action item issues

Requirements management tools can generate many of these metrics.

The last two metric types are determined through visual inspection and are subjective. Therefore, we only describe the first five of these.

The requirements volatility metric tracks the numbers of deletions, additions, and modifications to requirements over time. Clearly, these numbers must be tracked by tools. It is often the case that problems that emerge later in the project life cycle can be traceable to those requirements that have a relatively high volatility.

Traceability metrics include linkage statistics between requirements levels, such as those collected by ARM, and test coverage and span statistics. The test case coverage metric indicates that many requirements have test cases and the test span metric indicates how many requirements a test case covers. Trace linkage metrics can also be established between requirements and architectural and design elements. As the IEEE 830 qualities dictate, a high level of internal and external traceability is desirable.

Completeness metrics relate to the number of times that "placeholder" terms such as TBD are used, and could be related to the requirements decomposition levels as illustrated in Figure 4.8.

Defect density metrics indicate the number of requirements defects uncovered during a requirements inspection. These metrics are useful in predicting product and process volatility and interpreting other metrics.

Finally, requirements fault density indicates the number of requirements faults that are initially detected during test execution or posttest analysis. Requirements fault can be classified by criticality to obtain a number of different metrics. These metrics are used in determining the effectiveness of testing. Fault density metrics are used in predicting product/process volatility and quality and are essential in interpreting the meaning of other metrics (Costello and Liu 1995).

Exercises

8.1 What criteria should be used in choosing an appropriate requirements engineering tool?

8.2 Are there any drawbacks to using certain tools in requirements engineering activities?

8.3 What characteristics should you look for when selecting an open source tool?

8.4 How can tools enable distributed, global requirements engineering activities? What are the drawbacks in this regard?

8.5 If an environment does not currently engage in solid requirements engineering practices, should tools be introduced?

8.6 What sort of problems might you find through a traceability matrix that you might not see without one?

8.7 Download FreeMind and use it to brainstorm a mind map for your smart home system.

8.8 Construct a FitNesse test table for the requirement described in Section 7.2 of Appendix A.

*8.9 Using the Sud and Arthur dimensions for evaluating requirements management tools, conduct an assessment of five different commercial or open source tools using publicly available information.

References

Cleland-Huang, J., Settimi, R., Romanova, E., Berenbach, B., and Clark, S. (2007). Best practices for automated traceability, *IEEE Computer*, June: 27–35.

Costello, R.J. and Liu, D.-B. (1995). Metrics for requirements engineering, *Journal of Systems and Software*, 29(1): 39–63.

* This exercise is suitable for a small research assignment.

Eriksson, M., Morasi, H., Borstler, J., and Borg, K. (2005). The PLUSS toolkit—Extending telelogic DOORS and IBM-Rational Rose to support product line in use case modeling. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, Long Beach, CA, pp. 300–304.

FitNesse project website, www.fitnesse.org.

FreeMind project website http://freemind.sourceforge.net/wiki/index.php/Main_Page, last accessed 4 March 2013.

Gandhi, P., Haugen, N.C., Hill, M., and Watt, R. (2005). Creating a living specification using Fit documents. In *Agile 2005 Conference* (Agile 05), Washington, DC: IEEE CS Press, pp. 253–258.

Hammer, T.F., Huffman, L.L., Rosenberg, L.H., Wilson, W., and Hyatt, L. (1998). Doing requirements right the first time, *CROSSTALK The Journal of Defense Software Engineering*, (December): 20–25.

Heindl, M., Reinish, F., Biffl, S., and Egyed, A. (2006). Value-based selection of requirements engineering tool support, *32nd EUROMICRO Conference on Software Engineering and Advanced Applications. SEAA '06.* August, pp. 266–273.

Hull, E., Jackson, K., and Dick, J. (2011). Doors: A tool to manage requirements. In *Requirements Engineering*, New York: Springer, pp. 181–198.

Lee, S.-W., Gandhi, R.A., and Park, S. (2011). Requirement tracing. In *Encyclopedia of Software Engineering*, P. Laplante (Ed.), Boca Raton, FL: Taylor & Francis, 999–1011.

Lutz, R., Schäfer, S., and Diehl, S. (2012). Using mobile devices for collaborative requirements engineering. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (September), New York: ACM, pp. 298–301.

Maiden, N., Omo, O., Seyff, N., Grunbacher, P., and Mitteregger, K. (2007). Determining stakeholder needs in the workplace: How mobile technologies can help, *IEEE Software*, March/April: 46–52.

Open Source Requirements Management Tool (2013). Open source community, http://www.osrmt.com, last accessed March 2013.

Russell, S. and Creighton, O. (2011). Virtual world tools for requirements engineering. In *Multimedia and Enjoyable Requirements Engineering-Beyond Mere Descriptions and with More Fun and Games (MERE), 2011 Fourth International Workshop* (August), pp. 17–20.

Sud, R.R. and Arthur, J.D. (2003). Requirements Management Tools: A Quantitative Assessment. Technical Report TR-03-10, Computer Science. Blacksburg, VA: Virginia Polytechnic Institute and State University.

Volere Requirements Resources. (2013). Online at http://www.volere.co.uk/tools.htm, last accessed March 2013.

Yue, T., Briand, L.C., and Labiche, Y. (2011). A systematic review of transformation approaches between user requirements and analysis models, *Requirements Engineering*, 16(2): 75–99.