Chapter 1

Good practices for requirements engineering

The notion of best practices is debatable: who decides what is "best" and on what basis? One approach is to convene a body of industry experts to analyze projects from many organizations.

These experts seek out practices whose effective performance is associated with successful projects and which are performed poorly or not at all on failed projects. Through these means, the experts reach consensus on the activities that consistently yield superior results and label them best practices.

Table 2.1 lists more than 50 practices, grouped into 7 categories, that can help all development teams do a better job on their requirements activities. Several of the practices contribute to more than one category, but each practice appears only once in the table. Most of these practices

Table 1.1: Requirements engineering good practices.

Elicitation	Analysis	Specification	Validation
■ Define vision and scope ■ Identify user classes ■ Select product champions ■ Conduct focus groups ■ Identify user requirements ■ Identify system events and responses ■ Hold elicitation interviews ■ Hold facilitated elicitation workshops ■ Observe users performing their jobs ■ Distribute questionnaires ■ Perform document analysis ■ Examine problem reports ■ Reuse existing requirements	■ Model the application environment ■ Create prototypes ■ Analyze feasibility ■ Prioritize requirements ■ Create a data dictionary ■ Model the requirements ■ Analyze interfaces ■ Allocate requirements to subsystems	■ Adopt requirement document templates ■ Identify requirement origins ■ Uniquely label each requirement ■ Record business rules ■ Specify nonfunctional requirements	■ Review the requirements ■ Test the requirements ■ Define acceptance criteria ■ Simulate the requirements

Requirements management	Knowledge	Project management
■ Establish a change control process ■ Perform change impact analysis ■ Establish baselines and control versions of requirements sets ■ Maintain change history ■ Track requirements status ■ Track requirements issues ■ Maintain a requirements traceability matrix ■ Use a requirements management tool	■ Train business analysts ■ Educate stakeholders about requirements ■ Educate developers about application domain ■ Define a requirements engineering process ■ Create a glossary	■ Select an appropriate life cycle ■ Plan requirements approach ■ Estimate requirements effort ■ Base plans on requirements ■ Identify requirements decision makers ■ Renegotiate commitments ■ Manage requirements risks ■ Track requirements effort ■ Review past lessons learned

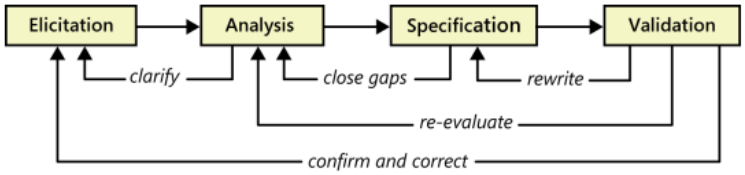
The people who perform or take a lead role in these practices will vary from practice to practice and from project to project. The business analyst (BA) will play a major role with many of them, but not every project has a BA. The product owner could perform some of the practices on an agile project. Still other practices are the purview of the project manager. Think about who the right people in your team are to lead or participate in the practices you select for your next project.

1.1 A requirements development process framework

As you saw in Chapter 1, "The essential software requirement," requirements development involves elicitation, analysis, specification, and validation. Don't expect to perform these activities in a simple linear, one-pass sequence, though. In practice, these activities are interwoven, incremental, and iterative, as shown

in Figure 2.1. "Progressive refinement of detail" is a key operating phrase for requirements development, moving from initial concepts of what is needed toward further precision of understanding and expression.

Figure 1.1: Requirements development is an iterative process.



- If you're the BA, you'll be asking customers questions, listening to what they say, and watching what they do (elicitation).
- You'll process this information to understand it, classify it in various categories, and relate the customer needs to possible software requirements (analysis). Your analysis might lead you to realize that you need to clarify some requirements, so you go back and do more elicitation.
- You'll then structure the customer input and derived requirements as written requirement statements and diagrams (specification). While writing requirements, you might need to go back and do some additional analysis to close gaps in your knowledge.
- Next, you'll ask some stakeholders to confirm that what you've captured is accurate and complete and to correct any errors (validation). You'll do all this for the set of requirements that are most important and most timely for beginning software development. Validation could lead you to rewrite some unclear requirements, revisit some of your analysis activities, or even have to go back and perform additional elicitation.

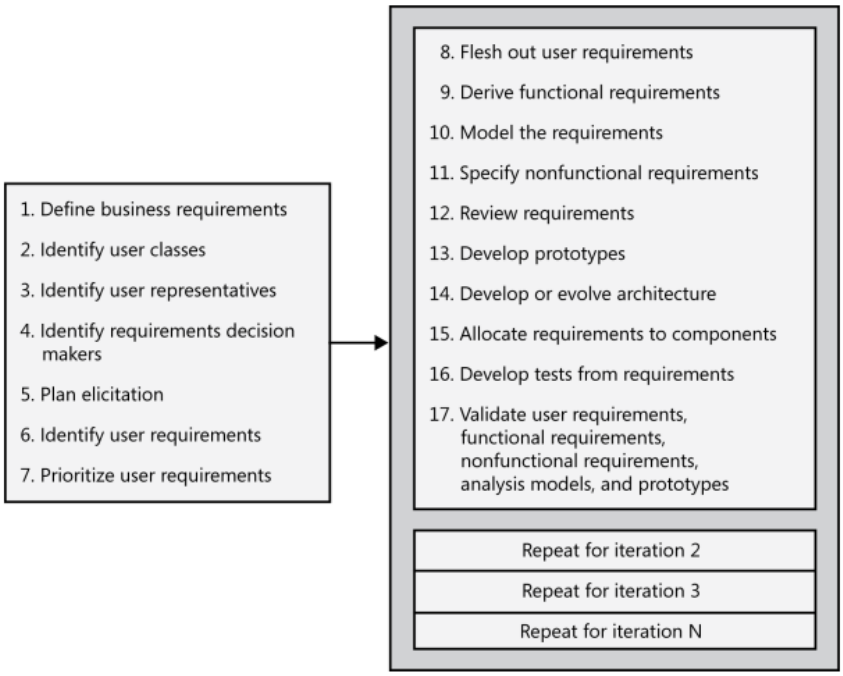
Then you'll move on to the next portion of the project and do it all again. This iterative process continues throughout requirements development and possibly—as with agile projects—throughout the full project duration.

Because of the diversity of software development projects and organizational cultures, there is no single, formulaic approach to requirements development. Figure 2.2 suggests a process framework for requirements development that will work, with sensible adjustments, for many projects.

- The business need or market opportunity is the predecessor for the process shown in Figure 2.2.
- These steps are generally performed approximately in numerical sequence, but the process is not strictly sequential.

- The first seven steps are typically performed once early in the project (although the team will need to revisit all of these activities periodically).
- The remaining steps are performed for each release or development iteration.
- Many of these activities can be performed iteratively, and they can be interwoven. For instance, you can perform steps 8, 9, and 10 in small chunks, performing a review (step 12) after each iteration.

Figure 1.2: A representative requirements development process.



Bibliography

[1] Bass, Len, Paul Clements, and Rick Kazman. 2013. Software Architecture in Practice, 3nd ed. Reading, MA: Addison-Wesley.

[2] Davis, Alan M. 1995. 201 Principles of Software Development. New York: McGraw-Hill.

[3] Sommerville, Ian, and Pete Sawyer. 1997. Requirements Engineering: A Good Practice Guide. Chichester, England: John Wiley & Sons Ltd.

[4] Box, George E. P., and Norman R. Draper. 1987. Empirical Model-Building and Response Surfaces. New York: John Wiley & Sons, Inc.

[5] Brown, Norm. 1996. "Industrial-Strength Management Strategies." IEEE Software 13(4):94-103.

[6] Kulak, Daryl, and Eamonn Guiney. 2004. Use Cases: Requirements in Context, 2nd ed. Boston: Addison-Wesley.

[7] Cohn, Mike. 2004. User Stories Applied: For Agile Software Development. Boston: Addison-Wesley.

[8] ISO/IEC/IEEE. 2011. "ISO/IEC/IEEE 29148:2011(E), Systems and software engineering—Life cycle processes—Requirements engineering." Geneva, Switzerland: International Organization for Standardization.

[9] Abran, Alain, James W. Moore, Pierre Bourque, and Robert Dupuis, eds. 2004. Guide to the Software Engineering Body of Knowledge, 2004 Version. Los Alamitos, CA: IEEE Computer Society Press.

[10] Beatty, Joy, and Anthony Chen. 2012. Visual Models for Software Requirements. Redmond, WA: Microsoft Press.

[11] Wiegers, Karl E. 2006. More About Software Requirements: Thorny Issues and Practical Advice. Redmond, WA: Microsoft Press.

[12] Wiegers, Karl E. 2007. Practical Project Initiation: A Handbook with Tools. Redmond, WA: Microsoft Press.

[13] Moore, Geoffrey A. 2002. Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. New York: HarperBusiness.

[14] Robertson, James, and Suzanne Robertson. 1994. Complete Systems Analysis: The Workbook, the Textbook, the Answers. New York: Dorset House Publishing.

[15] Robertson, Suzanne, and James Robertson. 2013. Mastering the Requirements Process: Getting Requirements Right, 3rd ed. Upper Saddle River, NJ: Addison-Wesley.

[16] Beatty, Joy, and Anthony Chen. 2012. Visual Models for Software Requirements. Redmond, WA: Microsoft Press.

[17] Nejmeh, Brian A., and Ian Thomas. 2002. "Business-Driven Product Planning Using Feature Vectors and Increments." IEEE Software 19(6):34-42.

[18] Booch, Grady, James Rumbaugh, and Ivar Jacobson. 1999. The Unified Modeling Language User Guide. Reading, MA: Addison-Wesley.

[19] Podeswa, Howard. 2009. The Business AnalystŚs Handbook. Boston: Course Technology.

[20] Armour, Frank, and Granville Miller. 2001. Advanced Use Case Modeling: Software Systems. Boston: Addison-Wesley.

[21] Business Rules Group. 2012. http://www.businessrulesgroup.org.

[22] von Halle, Barbara. 2002. Business Rules Applied: Building Better Systems Using the Business Rules Approach. New York: John Wiley & Sons, Inc.

[23] Ross, Ronald G. 1997. The Business Rule Book: Classifying, Defining, and Modeling Rules, Version 4.0, 2nd ed. Houston: Business Rule Solutions, LLC.

[24] Ross, Ronald G., and Gladys S. W. Lam. 2011. Building Business Solutions: Business Analysis with Business Rules. Houston: Business Rule Solutions, LLC.

[25] Ross, Ronald G. 2001. "The Business Rules Classification Scheme." DataToKnowledge Newsletter 29(5).

[26] Morgan, Tony. 2002. Business Rules and Information Systems: Aligning IT with Business Goals. Boston: Addison-Wesley.

[27] von Halle, Barbara, and Larry Goldberg. 2010. The Decision Model: A Business Logic Framework Linking Business and Technology. Boca Raton, FL: Auerbach Publications.

[28] Boyer, Jérôme, and Hafedh Mili. 2011. Agile Business Rule Development: Process, Architecture, and JRules Examples. Heidelberg, Germany: Springer.

[29] Gilb, Tom. 1988. Principles of Software Engineering Management. Harlow, England: Addison-Wesley.

[30] Ambler, Scott. 2005. The Elements of UML 2.0 Style. New York: Cambridge University Press.

[31] Withall, Stephen. 2007. Software Requirement Patterns. Redmond, WA: Microsoft Press.