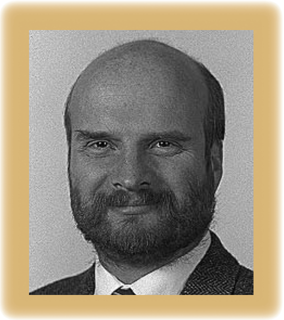
Editor: Neil Maiden ■ City University, London ■ n.a.m.maiden@city.ac.uk

10 Small Steps to Better Requirements

Ian Alexander

"The journey of a thousand miles begins with a single step." This Chinese proverb helps people focus on the present, rather than the unmanageable future.

Project teams can take several small, easy steps to improve requirements to the point where they're good enough. But every project is different. Your team might need to take steps that wouldn't be right in other situations.



The basic steps listed here, roughly in order, demand nothing more than a whiteboard or a pen and paper. Requirements management tools and specialized models can help later on.

Step 1: Mission and scope

If you don't know where you're going, you're not likely to end up there, said Forrest Gump. Is your mission clear? Write a short statement of what you want your project to achieve. Out of that will grow an understanding of the project's scope. The scope will develop as you more clearly understand exactly what achieving the mission entails.

A traditional context diagram is a good tool for describing scope in a simple outline. Unlike a use-case summary diagram, it indicates interfaces and the information or materials that must come in or out of them, rather than just roles (that is, UML actors). This becomes especially important when specifying physical systems where software is just a (large) component. Later, you can analyze each interface in detail.

Step 2: Stakeholders

Requirements come from people, not books. You can look in old system documents and read change requests, but you must still validate any candidate requirements you discover with people. Missing a group of stakeholders means your system could lack a whole chunk of functionality or other requirements.

Stakeholders are far more diverse than can be understood by listing their operational roles. Every system has a varied set of beneficiaries. The role of regulator is often critical. Stakeholders' negative opinions and actions can halt the project if not taken into account.

At the very least, you should list everyone with an interest in the system and consider what degree of involvement each person or group should have. A template might help you identify some stakeholders you'd otherwise overlook.

Step 3: Goals

Once you know who your stakeholders are, you can identify their goals. This can begin with a list or, perhaps better, a hierarchy (you could use an outliner, such as Microsoft Word's Outline View, or a mind-mapping tool).

Goals can be optimistic and even unrealizable—for example, a universally available product, a perfectly safe railway, or a ceaseless power source. In this way, they're unlike requirements,

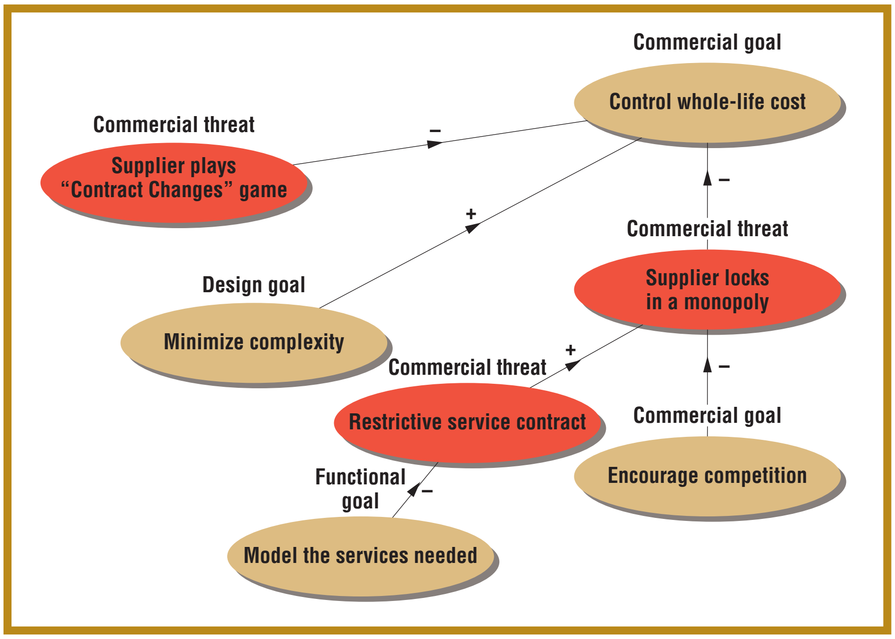


Figure 1. Analyzing goal and threat interactions quickly transforms vague targets into concrete functions. Here, it's applied to explore the core goals of a large public-service company that must upgrade its existing client-server system. The gold bubbles are desired goals; the red bubbles are threats.

which must be definitely realizable and verifiable.

Large initial goals are often vague qualities that stakeholders (for example, marketing or product management) want product users to experience. You can usually break such aspirations down into more concrete, realizable goals. For example, the goal "the car should feel luxurious" might be achieved with leather upholstery, air conditioning, soundproofing, walnut veneer, or hi-fi music. This second tier of goals suggests possible features and even software functions to help to meet the initial goal.

Step 4: Goal conflicts

Goals often conflict, but requirements must not. It's essential to discover and resolve any conflicts early on.

Some conflicts will be obvious from a plain list of goals; others are easier to see on a goal model (see figure 1). Draw each goal as a bubble; label it with its type (for example, commercial or functional). Draw an arrow marked with a plus sign (+) to show that one goal supports another; draw an arrow marked with a minus sign (–) to show a negative effect.

Step 5: Scenarios

Scenarios use the element of time to translate goals into connected stories. Isolated "shall" statements attempt to communicate needs as if they could be implemented independently, whereas connecting sequences of needs makes clear what dependencies exist.

At the very least, you should describe each operational situation as a brief story or numbered list of steps. Head each scenario with the name of the functional goal it implements. Write each step as a short statement in the form <role> <performs an action>.

When this is no longer sufficient, switch to writing use cases at a level of detail that suits your project. For example, Cockburn-style use cases include pre- and postconditions, making the transition to code far more controllable.

Step 6: Requirements

Scenarios don't cover everything. You will need techniques for interfaces, constraints (such as time and cost), and qualities (such as dependability and security).

Traditional "the system shall" statements are often convenient, not least because suppliers and standards expect them. Of course, you might think that the requirements consist of all the things listed here and more. However, not everyone shares that understanding.

Use cases do more than anything else to bridge the gap between the "now we do this" world of scenarios and the "you shall do that" world of traditional requirements. Neil Maiden has championed the systematic use of use cases to discover missing requirements ("Systematic Scenario Walkthroughs with ArtScene," *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, John Wiley & Sons, 2004).

Step 7: Justifications

If you note only what your system has to do and not why it has to do it, developers must guess what's really important. When resources start to run out or when planners see that demand will exceed available effort, project managers must cut or postpone something. A justification, like a priority, helps protect vital requirements from being dropped. It also helps developers understand how each requirement fits and why it's needed.

You can construct justifications in many ways—most simply with a few words explaining a requirement's purpose attached to individual requirements as attributes or listed separately and linked to requirements. You can construct more elaborate justifications as argumentation models, goals, risk models, or cost-benefit models.

Step 8: Assumptions

A system specification is a combination of things that you want to become true under system control (*requirements*) and things that you need to be true but can't control (*assumptions*).

Assumptions can include beliefs about existing systems, interfaces, competition, the market, the law, safety hazard mitigations, and so on. The most dangerous assumptions are tacit—the ones you didn't know you were making.

Like justifications, you can model assumptions as any kind of argumentation. The simplest approach is to state each assumption as a piece of text.

When planning a project, it can be valuable to examine the assumptions the

Further Reading

- **Mission and scope:** Dean Leffingwell and Don Widrig, *Managing Software Requirements,* Addison-Wesley, 2000.
  Suzanne Robertson and James Robertson, "Project Blastoff," *Mastering the Requirements Process,* Addison-Wesley, 1999.
- **Stakeholders:** Ian Alexander, "A Taxonomy of Stakeholders," *Int'l J. Tech. and Human Interaction,* vol. 1, no. 1, 2005, pp. 23–59.
  Ian Alexander and Suzanne Robertson, "Stakeholders without Tears," *IEEE Software,* vol. 21, no. 1, 2004, pp. 23–27.
- **Goals:** Alistair Sutcliffe, "Chapter 3: RE Tasks and Processes," *User-Centred Requirements Engineering,* Springer, 2002.
- **Goal conflicts:** Ian Alexander, "Misuse Cases: Use Cases with Hostile Intent," *IEEE Software,* vol. 20, no. 1, 2003, pp. 58–66.
- **Scenarios:** Alistair Cockburn, *Writing Effective Use Cases,* Addison-Wesley, 2001.
- **Requirements:** Ian Alexander and Richard Stevens, *Writing Better Requirements,* Addison-Wesley, 2002.
  Ian Alexander and Neil Maiden, "What Scenarios (Still) Aren't Good For," *Scenarios, Stories, Use Cases,* John Wiley & Sons, 2002.
- **Justifications:** Paul A. Kirschner, Simon J. Buckingham Shum, and Chad S. Carr, *Visualizing Argumentation,* Springer, 2003.
- **Assumptions:** James A. Dewar, *Assumption-Based Planning,* Cambridge University Press, 2002.
- **Agreed priorities:** Alan M. Davis, "The Art of Requirements Triage," *Computer,* vol. 36, no. 3, 2003, pp. 42–49.
- **Acceptance criteria:** Suzanne Robertson and James Robertson, "Fit Criteria," *Mastering the Requirements Process,* Addison-Wesley, 1999.

project depends on and to consider a course of action should any assumption fail and threaten the project's survival.

Step 9: Agreed priorities

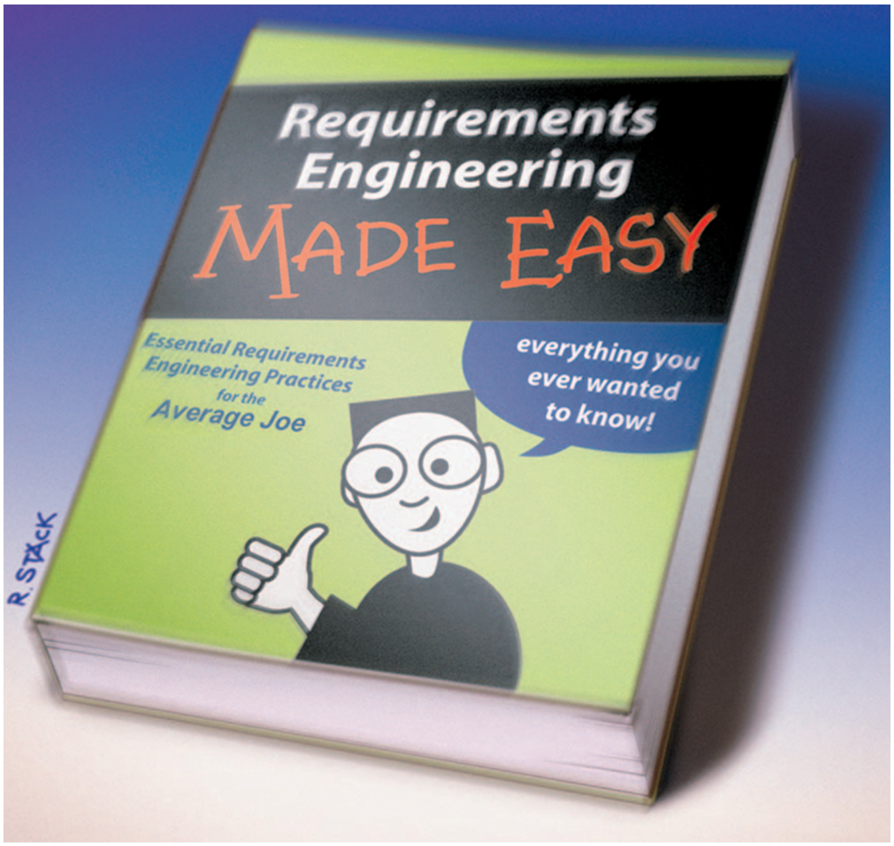
Some requirements are so obviously vital that you should just do them. Some are so obviously unnecessary that you should just drop them at once. You must prioritize the rest more closely. Dividing requirements into these three categories is a short, brutal process called *triage*.

Next, rank the remaining requirements into categories such as vital, desirable, nice to have, and luxury. You can do this by voting, reaching consensus among a panel, or allocating notional money.

Step 10: Acceptance criteria

Requirements don't end when you've completed the specifications. Rather, their work is just beginning—they guide development all the way to acceptance and service.

You should identify how you'll know when the system meets each requirement. This isn't a full description of each test case but criteria for deciding whether the system passes or fails against the requirement. Later, plan the test campaign on the basis of the scenarios (among other factors) and trace the test cases to the requirements to demonstrate coverage.


Requirements Engineering
MADE EASY
Essential Requirements Engineering Practices for the Average Joe
everything you ever wanted to know!
R. STACK

I can't guarantee that after taking these 10 steps, your requirements will be perfect. But if you follow them carefully, your requirements will improve.

Ian Alexander is an independent consultant specializing in requirements engineering for systems in the automotive, aerospace, transport, telecommunications, and public-service sectors. Contact him at iany@easynet.co.uk.