Habits of Effective Analysts[1]

Karl E. Wiegers
Process Impact
www.processimpact.com

Software managers sometimes assume that every skilled programmer is also proficient at interviewing customers and writing requirements specifications, without any training, resources or coaching. This isn't a reasonable assumption. Like testing, estimation and project management, requirements engineering has its own skill set and body of knowledge. Unfortunately, most computer science educational curricula emphasize programming-related knowledge over other software life cycle activities. In addition, many software practitioners—including myself—lack formal education in their field. Self-study and on-the-job learning often neglect softer skills such as those needed in requirements engineering.

The role of the requirements analyst (also called business analyst, systems analyst, requirements engineer or requirements manager) is critical to a software project. Many organizations expect developers or project managers to handle this vital function on their own. And even if a project team does include dedicated analysts, their skills might not be up to the task. Too often, I meet analysts who have had little training in how to perform their job and who have few books, articles or tools available to help them. Analysts who come from a user background may lack technical understanding, while those who migrated from the development world may not understand the user's business domain.

Every software organization should develop an experienced cadre of trained and knowledgeable requirements analysts, even though analysis might not be a full-time function on every project. An analyst provides a specialized capability that can make the difference between a project that succeeds and one that struggles. Here I describe several characteristics and practices of successful requirements analysts.

Bridge the Communication Gap

The analyst is a communication middleman, bridging the gap between vague customer notions and clear developer specifications. The analyst must first understand the user's actual needs and then define a set of functional requirements and quality goals that allow designers, implementers and testers to build and verify the system. If you aspire to be an effective analyst, become proficient in all forms of communication, including listening, speaking and writing. As you interact with executive project sponsors, marketing and user representatives, understand their objectives for the proposed system and their concerns about the business and the application. Learn and use the vocabulary of the application domain, rather than forcing your customers to understand computer jargon. Include business terms in a project glossary, which should become part of the requirements documentation.

Focus discussions with users on the tasks they must perform with the system—their use cases. Ask for examples of likely user goals, which will serve as the starting point for capturing scenarios you can use to develop accurate functional requirements. Don't be afraid to ask for

---

[1] This paper was originally published in *Software Development*, October 2000. It is reprinted (with modifications) with permission from *Software Development* magazine.

clarification; customers shouldn't expect every analyst to be an application domain expert. You might explain that you're not completely familiar with the customer's business and that is why you don't fully grasp what they are describing. This approach sometimes makes customers more willing to help because they can see that you're making a real effort to understand their problems.

We all think within the frame of our own experience and knowledge. Take the time to learn about your customer collaborators and understand how they prefer to communicate. Watch for assumptions that underlie either the users' expression of needs or your own thinking. Avoid imposing your personal filter of understanding on what you hear the customers say. Keep one of my axioms of software development in mind: The customer is not always right, but the customer always has a point. You must understand and respect those points, so they can be appropriately addressed in the product.

Try to understand the users' implicit expectations about the system's characteristics, such as performance, usability, efficiency and reliability. I've seen companies make products that fully met the customer's functional needs, only to discover that users hated the product because it didn't work like they thought it should. When users declare that the system must be "user-friendly," they have a mental image of what that means to them. As an analyst, your job is to understand the intent behind each such expectation, so you can translate something vague and subjective like "user-friendly" into goals the developer can satisfy. One way to approach this is to ask users what would constitute *unacceptable* performance, usability or reliability.

Requirements development should lead to an understanding, shared by the various project stakeholders, of the system that will address the customer's problem. The analyst is responsible for writing high-quality and well-organized requirements documents that clearly express this shared understanding. Writing documents that customer representatives can understand and verify, while unambiguously conveying precise functional and nonfunctional requirements to the developers, is a tightrope walk. A single requirements document might not meet both needs.

Typically, use case descriptions communicate information between analysts and users. Because use cases describe a user view of the system, users should understand them. However, use case descriptions alone often do not convey enough detail to the developer. One your tasks as an analyst is to derive from each use case the specific functional requirements that, when implemented, will enable users to perform the tasks described in the use case. This means you must be able to communicate effectively in both directions: with users (the task view) and with developers (the technical view). Work with representatives from both the producers and the consumers of key project documents like the software requirements specification to define appropriate templates for those documents. To make sure you've been understood, have user representatives, developers and testers review your documents.

Color Inside the Lines

Most projects that suffer from scope creep do so because the intended scope, the boundary between what is in and what is out, was never documented. Begin your exploration of a new system's requirements by defining the ultimate vision of what the product or application will be and do. Engage in a dialogue with the project's funding sponsor, marketing manager or product visionary early on to define the project's business requirements. You might suggest a suitable template for a vision and scope document or a marketing requirements document and work with those who hold the vision to help them describe it. Such high-level requirements documentation helps you answer this critical question whenever someone suggests a new product feature: "Is this feature in scope?" A sample vision and scope document template is available at www.processimpact.com/goodies.shtml.

Because the team probably won't implement the end result right off the bat, define the scope of the first release as a subset of the final product. Describe a growth path from the initial release toward realizing the ultimate vision through a series of staged releases. Also, document any known limitations or functionality that you don't intend to build but which some stakeholder might expect to find.

It's also important to focus the early requirements elicitation discussions on the tasks users need to accomplish with the system. Requirements discussions often center on fragments of functionality ("I need to be able to sort the list alphabetically"), quality characteristics ("this system has to be a lot more reliable than the old one"), or solution ideas ("then I select the state where I want to send the package from a drop-down list"). Don't discard these bits of information because they convey part of what the user has in mind. However, the process is more efficient if you encourage users to describe their goals in using the system, why they need the functionality or characteristics they describe, and the business tasks they're trying to accomplish. Use cases work well for this purpose. An understanding of user goals leads to the necessary functional requirements, which then leads to detailed user interface design.

Ask Revealing Questions

When working as an analyst, you'll need to actively facilitate discussions with users to pull out information that might otherwise go unstated. Ask questions to identify possible alternative ways a user might want to perform some task and related tasks that the user representatives didn't mention initially. If you hear a user say "the default should be …", he's probably describing the normal sequence of events for a use case. The phrase "the user should also have the option to …" suggests an alternative course for that use case.

Users naturally focus on the system's normal, expected behaviors. However, much code gets written to handle exceptions, so you should also search for possible error conditions that could arise and decide how the system should respond. If you don't describe exceptions during requirements elicitation, either the developers will make their best guess at how to handle them, or the system will simply fail when a user hits the error condition. It's a safe bet that system crashes aren't in the user's plans.

Rather than simply transcribing what customers say, a knowledgeable and creative analyst can suggest ideas and alternatives to users during elicitation. Sometimes users don't fully appreciate the capabilities that developers could provide, and they get excited when you suggest functionality that will really make the system useful. When users truly can't express what they need, perhaps you can watch them work and suggest ways to automate appropriate portions of the job. Analysts can often think out of the box that limits the creativity of people who are too close to the problem being solved. And be careful to avoid gold-plating, adding extra functionality that just seems cool or somehow appropriate. Look for opportunities to reuse functionality that is already available in another system. Users can sometimes adjust their requirements to exploit such reuse possibilities if the benefit is faster delivery of close-enough functionality that is already road-tested.

To function as an effective analyst, you must think at multiple levels of abstraction. You should be able to generalize from a specific need expressed by one user representative to define a set of related needs that will satisfy many members of that individual's user class. With experience, you'll become skilled in the art of asking questions that probe into the heart of each issue and clarify (or at least reveal) uncertainties, disagreements, assumptions and implicit expectations.

Prioritize Early and Often

Requirements development is an iterative and incremental process, proceeding from fuzzy notions to detailed specifications a layer at a time. If you're facilitating an elicitation workshop, keep the discussion focused on the right level of abstraction for that day's objectives. Don't let the participants get bogged down in excessive detail or premature system design. While it can be valuable to sketch out possible user interface screens or build prototypes to clarify the requirements, diving deeply into design too early can lead to a system that fails to meet the user's actual needs.

It's discouraging to realize at crunch-time that everything the team has left to do is essential, while they've already implemented some features that weren't really that important. Analysts must work with customers to define the priorities for requested product features, so the team can build the most critical functionality first. All customers will claim that their requirements should have the top priority. Your job as an analyst includes facilitating collaboration and negotiation between the various user classes and the developers to ensure that sensible priority decisions are made (see my article "First Things First: Prioritizing Requirements," Sept. 1999).

Early in requirements development, identify the various user classes that might contribute requirements. You'll need to understand which user classes are favored, which (if any) are disfavored, and which groups of users won't have input to the product's requirements. The favored user classes typically take precedence if you encounter conflicts in the requirements or priorities presented by different user classes.

Next, work with appropriate representatives of each user class to identify their major use cases. Performing a first-cut prioritization on the use cases will help you determine which ones to explore in detail early on. The top priority use cases are those that are the most important (capabilities the users really need) and the most urgent (capabilities the users need right away). The users might elect to implement only selected portions of certain use cases in the initial release, leaving refinements for later. Alternatively, they might opt to initially implement the full functionality of a small set of use cases. Understanding the logical dependencies among the requirements will allow you to determine whether some high-priority requirements should be delayed because of architectural constraints that demand that other functionality be built first.

Create a Collaborative Environment

Software development is often characterized by strained relationships among developers, users, managers and marketing. The parties may not trust each other's motivations or appreciate each other's needs and constraints. In reality, though, the producers and customers of a software product share some common objectives. For information systems development, all parties work for the same company and benefit from improvements to the corporate bottom line. For commercial products, developers and marketing should strive to meet the purchasing customers' needs, so they will buy more products and rave to their friends. And contract developers should try to make the client happy to get repeat business. A win/win outcome means customers are delighted with the product, the developing organization is happy with the business outcomes, and the development team members are proud of the good work they did on a challenging and rewarding project.

Achieving a win/win outcome requires honesty. Sharing all relevant information among the stakeholders and telling the truth without blame or judgment promotes free and informed choices. Such a ideal environment isn't always achievable. In fact, none of my suggestions are likely to work if you're dealing with truly unreasonable people.

Defining business requirements early in the project will clarify the prospective benefits for both customers and the developing organization. The participants also need to be honest about functionality costs and project constraints. If you think the customer's cost or schedule expectations are unrealistic, tell them so and explain your reasoning. Considering the costs will help the stakeholders make sensible business decisions to achieve the maximum value within the existing resource, time and technology constraints.

It's not unusual for an analyst to solicit input from users only to be told, "I don't have time to talk to you. You should know what I need." However, software success is most likely when the analyst can forge an effective collaborative relationship with key customer representatives (see my article "Requirements and the Software Customer," Dec. 1999). Identify individuals who could serve as the voice of the customer for each of your system's user classes, then engage them in dialogs about their needs.

User representatives might hesitate to participate in requirements exploration until they know exactly what you expect from them. Write down the contributions you would like to get from your customer collaborators and negotiate an appropriate level of commitment from each one. The vision and scope document will help you identify the right users to talk to. It also gives the user representatives a clear understanding of what the project is trying to achieve.

Insufficient user involvement is well established as a leading cause of software project failure. Point this out to recalcitrant users or managers who don't want to spend time on requirements discussions. Remind your customers of problems they've experienced on previous projects that can be attributed to inadequate user involvement. Nearly every organization has horror stories of new systems that didn't satisfy user needs, failed to meet unstated usability or performance expectations, or duplicated the shortcomings of the preceding systems. You can't afford to keep rebuilding or discarding systems that don't measure up because the user needs weren't sufficiently understood. If customers won't commit to reaching a shared understanding of their requirements, the development organization might be better off avoiding the project. Otherwise, the outcome might well be lose/lose.

Hone Your Skills

The requirements analyst provides the essential function of bridging the understanding and perspective gap that lies between customers and developers. A competent analyst must combine communication, facilitation and interpersonal skills with some technical and business domain knowledge. Even a dynamite programmer or a system-savvy user needs suitable preparation before acting as an analyst. The following capabilities are particularly important:

- facilitation techniques, to lead elicitation workshops;
- interviewing techniques, to talk with individuals and groups about their needs;
- listening skills, to understand what people say and to detect what they might be hesitant to say;
- writing skills, to communicate information effectively to users, managers and technical staff;
- organizational skills, to make sense of the vast array of information gathered during elicitation and analysis;
- interpersonal skills, to help negotiate priorities and resolve conflicts among project stakeholders; domain knowledge, to have credibility with user representatives and converse effectively with them; and
- modeling skills, to represent requirements information in graphical forms that augment textual representations in natural language

An effective requirements analyst has a rich toolkit of techniques available and knows when—and when not—to apply each. My book *Software Requirements* (Microsoft Press, 1999) describes more than 40 requirements engineering good practices. Tools ranging from the venerable flowchart, through structured analysis models (context diagram, data flow diagram and the like), to contemporary UML notations should be part of every analyst's repertoire. The effective analyst can spot a roadblock to understanding and select appropriate tools to get around the roadblock.

There is no substitute for experience. One of my consulting clients discovered that they could inspect requirements specifications written by experienced analysts twice as fast as those written by novices because they contained fewer defects. In contrast, an organization that asked each developer to write the requirements for the system components for which he was responsible encountered wildly divergent styles, organization and quality. This made it difficult for developers to review and use each other's specifications. A third organization appointed as analysts several developers for whom English was not the native language, while yet another expected its users to write up their own requirements. It's hard enough to write good requirements when you do it for a living, let alone if you do it just once in a while or in a language in which you aren't completely proficient.

Requirements for a software product aren't just lying around waiting for someone wearing a hat labeled "analyst" to collect them. At best, requirements exist in the minds of users, visionaries and developers, from which they must be gently extracted and massaged into a usable form. Often, they need to be discovered with guidance from a talented analyst, who helps users understand what they really need to meet their business needs and helps developers satisfy those needs. Few project roles are more difficult than that of requirements analyst. Few are more critical.