Communicating Requirements To Testing

You're Going To Be In Pictures!

By Trevor Atkins **October 29, 2008**

Communication is the process of attempting to convey information from a sender to a receiver with the use of a medium. Effective communication requires that all parties have a basis of commonality in which a shared understanding on a particular subject can be achieved.

Human communication was revolutionized long ago with speech, which greatly facilitated the communication of information and knowledge. Information communicated via spoken language became increasingly rich, and allowed humans to work and advance together much more quickly than was previously possible. Speech meant easier coordination, cooperation, technological progress, and development of complex, abstract concepts.

However, speech depends on the imperfect tool of human memory, through which information can become corrupted with each re-telling or even lost entirely to the passage of time, and there is a limit to how much information can be remembered by the receiver in a single 'communication'. Finally, the information is stored in the fragile form of the human body – with the sudden death of a 'wise man' or elder, a tribal group could abruptly lose generations of accumulated knowledge.

The imperfection of speech eventually resulted in the development of new forms of communication, improving the range at which people could communicate as well as increasing the volume and longevity of the information involved. These advancements were based on the development of writing in which the representation of concepts through pictographs and symbols was the key early driver.

WHITEPAPER

How do you know what a system is supposed to do and what it is not supposed to do? Formal requirements are intended to create an easily validated, maintainable and comprehensive set of documents communicating the system's planned functional scope in terms of tasks and behaviours.

It is well understood that a higher quality product demands a higher upfront investment. However, compromises regarding quality versus cost are made every day and tight project budgets and short timelines often greatly reduce the interest in formalized documentation, requirements or otherwise.

> *"The communication of functional requirements and specifications is the most difficult, critical, and error-prone task in IT projects. Research has shown that projects that proceed to the construction and coding phase with missing or wrong functional requirements and specifications are almost certain to fail."* – Bill Walton, "A Systematic Approach for More Effective Communication of Functional Requirements and Specifications"

A balanced approach that maximizes the contribution of the organization's available resources within the project constraints is required. The following proven light-weight solution tailored to your timeframes and resource realities can allow testing to, in the absence of complete formal requirements, capture critical information about the system without slowing development. The resulting artifacts can also serve as a direct input into testing activities – avoiding significant duplication of effort and possible confusions.

Who Needs Requirements?

In the fast-paced changing world of software development there is a continuous challenge to communicate the expectations for the system and its internally and externally facing behaviours. Formal, documented requirements often suffer because of the challenges in keeping up with short iterative project life-cycles, continually evolving product scope and customer demands, and uncertain or changing screens and interfaces.

> *"Industry data suggests that approximately 50 percent of product defects originate in the requirements. Perhaps 80 percent of the rework effort on a development project can be traced to requirements defects. Anything you can do to prevent requirements errors from propagating downstream will save you time and money."* – Karl Wiegers, "Inspecting Requirements"

And yet, requirements information is a primary input to the majority of the stakeholders' project activities. Consider the following example stakeholder groups:

- **Marketing:** Promotion of system capabilities, competitive comparisons
- **Customer:** Scope definition, description of business needs and users / roles, definition of acceptance criteria
- **Business Analysis:** Elicitation and capture of business logic and tasks from the customer and other stakeholders
- **Project Management:** Scope management, risk planning, effort estimation, project goal setting, project and resource planning
- **Development:** Design and implementation
- **Testing:** Verification and validation
- **Technical Writing:** Creation of user manuals and tutorials

It is evident that all stakeholders have a vested interest to collectively optimize agreement, while simultaneously minimizing risk and rework costs on a given project. But, without clearly stated requirements, testing, specifically, is unable to contribute effectively to these goals or to perform its job of both verifying that the system was implemented correctly and validating that the correct system was implemented.

This impairment also extends to responding to the need to accurately measure and report the test coverage of the system. Without documented requirements and the associated dependent artifacts, it is virtually impossible to properly answer questions such as the following:

- Have all the requirements been implemented?
- Have tests been created to verify each of the requirements?
- Have all the requirements been tested for this build / release?
- Which areas of the system are stable and which are not?

Unless requirements can be traced accurately through the related project artifacts, such as business requirements, use cases, functional requirements, design specifications, code, test results, and defects, it is impossible to ensure that the system is being built for the intended purposes and tested adequately – keeping the project on track and avoiding expensive rework later.

! Requirements traceability is also beneficial for risk analysis when a requirement needs to be changed.

"Sometimes developers or project managers agree to make suggested changes without carefully thinking through the implications. The change might turn out to be more complex than anticipated, take longer than promised, be technically or economically infeasible, or conflict with other requirements." – Karl Wiegers, "Karl Wiegers Describes Ten Requirements Traps to Avoid"

Working With The Existing Requirements

Most projects will have requirements information in some form or another whether it is formal specifications, wireframes, mock-ups, prototypes, dialog maps, content sheets, previous system releases, user manuals, or even stored in the minds of subject matter experts (SME's). And it is very likely that the customer has contributed to the requirements by describing what they want from the system in terms of their business needs.

In general, many of these requirements sources by themselves will not provide enough detail for developers to code or for testing to test without making interpretations, assumptions, or guesses. Use cases and functional requirements are needed to provide the data flow details and the functional capabilities of each component. And, as the project lifecycle progress, more artifacts will be included – the design specification, the code, and the tests which will link to the use cases and the functional requirements, and through them, to the customer's business requirements.

Writing Quality Requirements

It is common to have some portion of the requirements information documented in a formal manner for consumption by all stakeholders including testing.

"The criticality of correct, complete, testable requirements is a fundamental tenet of software engineering. The success of a project, both functionally and financially, is directly affected by the quality of the requirements." – Theodore F. Hammer, Linda H. Rosenberg, et al., "Doing Requirements Right the First Time!"

Formally written requirements are typically described in natural language such that both the customer and vendor or project team can understand them. However, many words and phrases have meanings that can be interpreted based on the context in which they are used. Requirements described in this form can have several severe problems including: ambiguity, inaccuracy and inconsistency.

Frequently, the impact of these problems surface late in the project at the acceptance phase as discrepancies are realized between what was built and what the customer thought was being built – a clear case of the lack of quality requirements directly causing customer dissatisfaction.

> *"The ultimate symptom of vague requirements is that developers have to ask the author, analyst or customers many questions, or they have to guess about what is really intended.*

> *The extent of this guessing game might not be recognized until the project is far along and implementation has diverged from what is really required. At this point, expensive rework may be needed to bring things back into alignment."* – Karl Wiegers, "Karl Wiegers Describes Ten Requirements Traps to Avoid"

Involving testing early in the project lifecycle means the chance to review requirements for important quality attributes, ask questions, and get issues resolved before they become much more expensive problems in the code.

Using a style guide when both writing and reviewing requirements can help alleviate the problems in this area. With a well-defined style guide that addresses the quality attributes, such as those identified by the NASA Goddard Space Flight Center's (GSFC) Software Assurance Technology Center (SATC), metrics can be easily employed to reveal the strengths and weaknesses of the formal requirements documentation while there is still time to fix any issues.

! The SATC developed the Automated Requirements Measurement (ARM) tool which can be used to assess the quality of a project's requirements documents easily and on an on-going basis during the life of the documents. The ARM tool searches the requirements document for terms the SATC has identified as quality indicators.

For more information on the NASA SATC quality attributes and the ARM tool, visit: http://satc.gsfc.nasa.gov/tools/arm/

Similar to a development coding standard, a requirements style guide outlines when and where terms and structures must and may alternately be used, thereby helping maintain control over ambiguity, consistency, correctness, and completeness.

What Requirements Are Missing?

The upfront time that it takes to document formal requirements properly and completely is often perceived as a barrier to avoid even though it is easy to recognize that the proven savings downstream more than return the investment.

For example, if are you shipping software applications under significant time-to-market pressures, you likely lack many formal requirements because of the huge pressures to show tangible progress from the end-users' points of view. You don't want to slow down development or testing by having to create detailed documentation. At the same time, the test effort needs to be useful and measurable or the customer will be the one reporting all the issues whether they are code or specification related. How, with limited formal requirements, can this system be tested to achieve adequate effective coverage and overall stability of its functionality?

Testing can greatly benefit from engaging in up-front information gathering to increase understanding of how the software is intended to work and in which situations, especially when formal requirements may be few to none.

In the rushed atmosphere of 'we need to start coding and we need to start now', it is hard to convince anyone to take the time to perform 'understood' formal documentation activities, let alone develop and apply a new documentation technique.

! A Standish Group CHAOS study of over 8,300 IT projects found that more than 50% were "challenged" with reduced functionality being delivered over-budget and beyond the estimated schedule. The main reasons were a lack of user input and incomplete and changing requirements.

However, the following will outline a solution approach to capture the very necessary requirements information in a rapid but sophisticated manner that will feed directly into the needs of testing but also potentially facilitate the needs of the other stakeholders. And this approach will leverage the test resources of your project team while they await implemented functionality to be delivered for testing.

Pictures for Testing

When you have minimal or out-of-date requirements, a first step in capturing a useful understanding of the system to be tested and filling in the gaps regarding its purpose and functions, is to think in pictures.

Every picture helps tell a story and stories or scenarios form a basis for analysis and testing.

> *"Imagery is the most fundamental language we have. Everything you do, the mind processes through images."*
> – Dennis Gersten, M.D. published in Atlantis, a bi-monthly imagery newsletter

The Unified Modeling Language (UML), which is a standard for specifying, visualizing, and documenting the artifacts of software systems, can be employed to help provide these pictures. However, there are less formal types of notations you can use to put together straightforward diagrams, such as activity flowcharts, data flow diagrams, state diagrams, and sequence diagrams, that can be more useful for meeting your project's testing needs.

> *"Pictures can pack a great deal of information into a small space. They help us to see connections that mere words cannot."* – Elizabeth Hendrickson, "A Picture's Worth a Thousand Words"

Using diagrams can be very effective to visualize the software, not only for the tester but for the whole project team and as long as you can capture the information you need to generate comprehensive, traceable test ideas, it doesn't matter what notation you use.

The First Sketch

What do you know about the system to be tested? Is it a client-server application? What are the major subsystems? Is it web-based? Is there a database? Are there external files being accessed or written? Are there clusters? What are the major tasks the system is supposed to perform?

Start by asking end users, project managers, developers, and other stakeholders these basic 'W5H' questions to assist in clarifying the system's scope:

- Why is the system being built?
- Who are the end users of the system?
- What are the tasks to be performed?
- When are there interactions between modules? With other systems?
- Where will the system be deployed?
- How is the system being built? (architecture)

Capture this information in an annotated diagram of the overall system.

The following is an example illustration of what part of this initial picture could look like for a corporate telephone system:

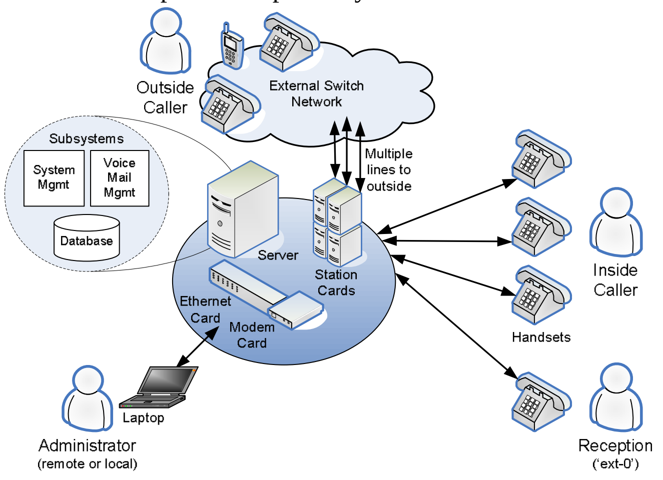

Figure 1: First depiction of a corporate telephone system

Adding the Details

Next, drill-down on the depiction of the main modules and the list of tasks to address when, why, and how they interact. From this point, you can proceed to create more detailed diagrams for the functionality and the workflows within each module and those that span multiple modules.

When capturing these more detailed views of the system, also capture the following information for each module, workflow, and the system overall:

- State which are the primary tasks of the system
- Describe sequences in which tasks are normally performed
- Describe the user roles / responsibilities and what is common system usage for each on a periodic basis, eg: daily, monthly, or yearly
- Point out system limitations, eg: things that the system cannot or is not supposed to do
- Summarize the system's environment, eg: target platforms and other systems with which it must interface or co-exist
- Outline or list functionality that is not included as part of the workflows – use checklists

Once you have outlined a rough view of how the system is supposed to behave, work with end users and applicable stakeholders to review your diagrams and short descriptions to help in defining the associated scenarios in more detail. This simple collaboration will not only help you understand what the customer is expecting but will also allow you to rapidly disseminate and validate the information you have captured so far with the project team.

At the same time, describing the tasks and subtasks in detail will lead directly test scenarios and analyzing the relationships among the modules will help determine the important inputs for the overall testing strategy.

Tools to Develop Your Pictures

Flowcharts and state diagrams are powerful visualization tools for capturing the behaviours and functionality of a system, establishing path coverage and generating ideas for error path or negative testing. Narrative user scenarios can be used to complement and expand upon what is provided in the diagrammatic representation. Adding checklists and matrices will further enable testing to be able to rapidly document the functionality to be tested and to be able to measure or assess the progress of that testing easily.

! The flowchart, state diagram and user scenario examples given below are presented as 'first depictions'. As such they are not necessarily complete or even correct.

The examples illustrate that the rapid collection of information can be made through a combination of pictorial and narrative approaches that facilitate easy review and discussion leading quickly to traceable test ideas.

Flowcharts

A flowchart is a common pictorial representation of a process or task, describing the logical flow of decision points and activities. Flowcharts are useful for defining the paths you want to verify or to force from an error condition.

The following is an example illustration of what one annotated scenario for a corporate telephone system could look like as a flowchart:

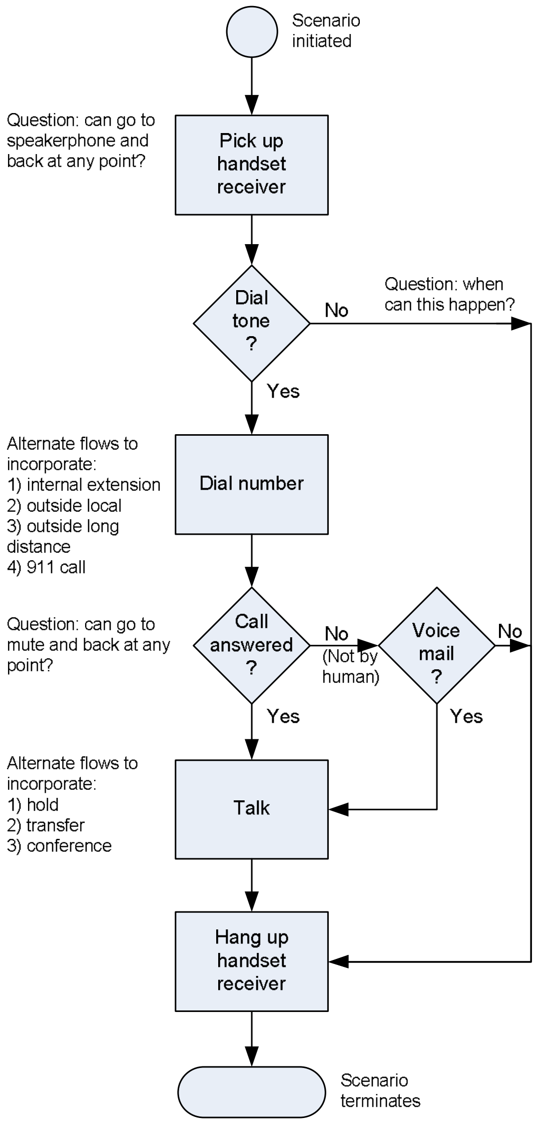

Figure 2: First depiction of the 'Making a Call – Originator Is Inside Caller' scenario

Flowcharts provide an excellent form of rapid, largely visual, documentation that makes it easy to examine how various steps in a process or task should connect together.

State Diagrams
Another option to capture system behaviour is through the use of state diagrams. State diagrams describe all of the possible states of an object as events occur and the conditions for transitions between those states.

The following is an example illustration of what states the handset of a corporate telephone system could have:

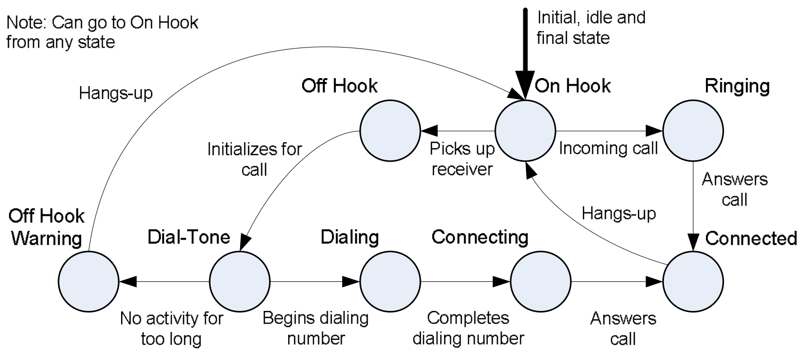

Figure 3: First depiction of the states of the system handset

All state diagrams begin with an initial state of the object. Transition conditions based on activities determine the next state of the object. State diagrams can help reveal states in the system that are not obvious to or are hidden from the actual users.

User Scenarios
User scenarios define a sequence of actions completed by a system or user that provides a recognizable result to the user. Like formal requirements, a user scenario is written in natural language that draws from a common glossary. The user scenario will have the basic or typical flow of events (the 'must have' functionality) and the alternate flows.

Example user scenario: The following provides an example outline of a user scenario:

Name: Making a Call – Originator Is Inside Caller
Actor(s): Inside Caller, (Outside Caller, Receptionist)
Basic Flow:

1.0 Inside Caller picks up receiver of handset
2.0 Inside Caller waits for dial-tone
3.0 Inside Caller dials a phone number on handset keypad. In the case of an outside destination the Receptionist's handset will show a new line is now in use.
4.0 The call is answered by the destination
5.0 Inside Caller and the destination are able to speak to each other
6.0 Inside Caller hangs up receiver of handset

Alternate Flows:

3.1 Inside Caller may dial '0' to reach the Receptionist station handset
3.2 Inside Caller may dial a 3-digit extension starting with the digit '1' to reach any other active extension within the corporate telephone system connected with a station handset
  3.2.1 If Inside Caller dials an inactive extension an error message will be given

3.3 Inside Caller can dial an outside phone number to reach Outside Caller by first pressing the 'outgoing' button on the handset keypad
  3.3.1 If Inside Caller dials a long distance number as the outside phone number, a 4 digit passcode will be required to complete the dialing
3.4 Inside Caller can connect to emergency services by dialing the 3 digits '911'

Pre and Post Conditions should also be included when significant.

Benefits of creating user scenarios:

- Easy for the owner of the functionality to tell/draw the story about how it is supposed to work
- System entities and user types are identified
- Allows for easy review and ability to fill in the gaps or update as things change
- Provides early testing or validation of architecture, design, and working demos
- Provides systematic step-by-step description of the systems' services
- Easy to expand the steps into test scenarios or test cases

User scenarios quickly provide a clearer picture of what the customer is expecting the product to accomplish. Employing these user scenarios can reduce ambiguity and vagueness in the development process and can, in turn, be used to create very specific test ideas and validate the functionality, boundaries, and error handling of a program.

Creating user scenarios can be kick-started by simply drawing a flowchart of the basic and alternate flows through the system. This exercise rapidly identifies the areas for later testing, outstanding questions, and even design issues.

Checklists

Are there common types of tasks that can be performed throughout the system? Are there tests so straightforward that the usual scenario description and reproduction steps are not necessary (ie: inspection type tests)?

Checklists are useful tools to ensure test coverage of these common tasks, such as:

- User interface standards
- Data entry (eg: Data type error and boundary tests)
- Certain features (eg: Search functionality tests)

Benefits of creating checklists:

- Easy to maintain as things change
- Easy to add to and improve as time passes
- Captures the tests being performed in a central location for consistency and repeatability of execution across the test team and the test cycles.

Used in conjunction with user scenarios and flowcharts, checklists are part of a powerful combination of light-weight test planning techniques.

! James Bach of Satisfice.com provides a number of whitepapers and articles on 'Exploratory Testing' wherein he has a set of mnemonics and heuristics in his toolkit. One of these mnemonics is SFDPO where the letters stand for Structure, Function, Data, Platform, and Operations. Using rules and checklists such as these allow you to quickly focus your test idea generation and ensure that you have systematically visited the major aspects of the product.

Matrices (2D+ Checklists)
Test matrices are ideal for tracking the execution of a series of tests when there are multiple dimensions to a checklist, such as roles, environments, configurations, and/or versions (builds) of the application.

Benefits of using test matrices:

- Easy to maintain as priorities change and functionality becomes available during the project lifecycle
- Simple to prioritize the functional areas and the tests in each area
- Ensure coverage of the appropriate user roles and their permissions to functional areas
- Visible progress tracking of the test effort for each build / release across the test configurations
- Easy to identify problem areas or environments as the project proceeds

Test matrices can include checklists and user scenarios along with other test suites and specific individual tests to provide a clear picture of what has been done and how much is left to do.

Conclusion

So why should testing go to the trouble of searching out the missing requirements or taming the wild, ambiguous ones? Addressing the aspects of quality (or lack thereof) on a project leads to working smarter rather than harder, better products, more satisfied customers, and therefore higher profits and a successful company. But, you will never have enough time to do everything the 'right' way.

However, if you thoughtfully invest some time to better understand the software to be implemented and tested you can not only improve your actual testing activities, but also help improve the entire project team's agreed understanding of the product through the increased communication – thereby significantly improving all aspects of product and project quality.

"When all the pieces come together - the right people, the right processes, the right time, the right techniques, the right focus - then we can achieve truly impressive returns on our testing investment. Significant reductions in post-release costs are ours for the taking with good testing. In cost of quality parlance, we invest in upfront costs of conformance (testing and quality assurance) to reduce the downstream costs of nonconformance (maintenance costs and other intangibles associated with field failures)." – Rex Black, "Investing in Software Testing"

Flowcharts and state diagrams provide similar and at times complementary methods for visualizing, or picturing, the core information to be captured in a user scenario. Through the rapid process of creating these diagrams and subsequently reviewing and clarifying them and their stories with stakeholders, test ideas will easily come to the fore for later execution.

Of course, even with the best tools and techniques, to be most effective the test team's efforts must start early in the project and involve all appropriate stakeholders and participants. So when you go to implement the solution outlined above, remember to involve subject matter experts and other stakeholders in the creation and review of the requirements information you collect; use pictures and checklists to facilitate communication, accelerate understanding and confirm scope and priorities.

About the Author

Trevor Atkins is Principal Consultant with Silverpath Technologies (www.silverpath.com). Most recently he was a Regional Director of Quality Services with UST Global and before that he was a founder and the Vice-President Operations of QA Labs, the largest independent software testing company in Canada.

After obtaining a degree of Applied Science in Electrical Engineering from the University of British Columbia, Trevor has been involved in all aspects of hundreds of successful software projects for the last 12+ years, and is dedicated to the design and improvement of quality processes for use across projects and organizations.

About Silverpath Technologies Inc.

Silverpath Technologies provides results-centric consulting and training services targeted at improving the effectiveness and efficiency of quality and testing activities across the software development lifecycle. Visit http://www.silverpath.com for more information.