Editors: Dave Thomas and Andy Hunt ■ The Pragmatic Programmers
dave@pragmaticprogrammer.com ■ andy@pragmaticprogrammer.com

Nurturing Requirements

Dave Thomas and Andy Hunt

As this issue's focus is requirements engineering, we thought we might surprise our editor this month and actually write an on-topic article. Given that this is the Construction column, though, are we going to be able to pull it off? Stay tuned.

As a starting point, let's pull a quote from the previous issue of *IEEE Software*. In their column "Understanding Project Sociology by Modeling Stakeholders" (Requirements, Jan./Feb. 2004), Ian Alexander and Suzanne Robertson describe the reactions of developers who are asked what they mean when they say "We have a problem with our stakeholders." One of the responses was

> *The stakeholders ... don't have the skills necessary to participate in gathering requirements. They describe solutions, rather than requirements, and they change their minds.*

Users, focused on solutions? Shameful! Users, changing their minds? Never!

It's hard to sympathize with developers who express this kind of frustration. In the real world, stakeholders are interested in solutions, not some abstract developer-centric set of requirements. And users most definitely change their minds: perhaps they got it wrong, or perhaps the world just changed.

But the developers quoted by Alexander and Robertson are not alone. A kind of mass delusion seems to persist in the development community. Many people think that there are such things as requirements.

No Virginia, there are no requirements

The fundamental problem here is that folks believe that underlying every project there's some absolute, discoverable set of requirements. If only we could find them, and then build accurately against them, we'd produce a perfect solution. The requirements would be a kind of map, leading us from where we are to our project's final destination.

This is a wonderful dream, but it's unlikely to be true.

Part of the problem lies in defining what we actually mean by *requirements*. The dictionary doesn't help much: requirements are the things we need or wish for. As a community, we can deal with the *need* part of this definition. These are a project's constraints. If I'm interfacing to a payment gateway, I'm constrained to use that gateway's communications protocols and to obey the transactional rules of the financial institutions with whom my code interacts. If I'm writing autopilot software for the next-generation business jet, I have a considerable list of constraints, imposed by the laws of both nature and man.

The nice thing about constraints is that I can test the eventual application against them. I can generate traffic from my application into a mocked-up payment gateway and check the messages' format and content. I can generate both valid and exceptional responses and verify that my application handles them as specified by the protocol. I can do the same with the autopilot software—first in a simulator, and later in a real plane. (There's an apocryphal story of a UK development team in the 1970s who wrote software to help a particular helicopter hover automatically. Part of the acceptance test was for the team to ride in the helicopter while the pilot released the controls and their software flew the plane. Seems to us that this kind of testing would focus the mind tremendously.)

But there's the second side of requirements: the wishing side. This is where users get to talk about solutions and needs.

For a typical business application, the wishing side is typically more prominent than the constraint side. Users typically say, "I need a report showing the sales breakdown by product." Ask them, "OK, so exactly what are your requirements?" and they'll look at you funny and repeat, "I need a report showing the sales breakdown by product." You can dig deeper and deeper, and they may start giving you more information, but in reality that information is typically of doubtful value: force a business user into a corner, and they'll invent something on the spot just to stop you annoying them. Are they being capricious? Not at all. In their world view, "I need a report…" is a perfectly good requirement. They see no need to give you a layout, or to define the colors or fonts, or tell you what database tables to query. These are implementation details, not fundamental requirements.

So, our users can't always give us what we want when we ask for firm requirements. Often they don't know the details. And often the details change along the way. What are we to do?

Constructing requirements

This series of columns is called Software *Construction*. It's a metaphor that the software industry has embraced when describing what we do. (Andy and I actually have some serious misgivings about the term, but that's the subject of a later article.) So how do other construction-oriented industries deal with requirements? Think about the process of building a custom house.

As the owner-to-be, you get to specify what you want: number of rooms, number of stories, overall style, and so on. These are the constraints side of your requirements. When the house is built, you'll be able to test them (does the property have the requisite number of rooms and stories, for example). But you have other hopes for the house as well. You want it to have the right feel. You want it to flow well. These intangibles are hard to specify precisely. So how does the architect deal with this? They iterate. When you first start the process, they sit down with you and sketch out some ideas. The next time you see them, these sketches may have turned into a set of high-level plans, along with an elevation or two to give you a feel for the property. You may spend a couple of meetings iterating these drawings until things get close. Then, if your budget extends that far, you may even get to see a model of your final house.

Anyone can build castles out of clouds, but once the mortar hits the bricks, you soon start to realize what's important and what isn't.

Next, the plans go to the builder, who starts framing the property. At this stage, you still have some control. As the boards start to go up, and as you walk around the site, you realize you don't like a doorway where it is, or that some dead space under the stairs would make a great place to store the kids' toys. These changes clearly cost you some time and money, but they're easier now than when the house is complete. Because you, the user, were involved throughout the process, you were able to make many small adjustments to the house as it progressed through conception, design, and construction. Hopefully, the final product will be closer to your dreams than it would have been had you met once with the architect while she captured your requirements and then come back eight months later to see the resulting property.

The same kind of continuous user involvement and constant iteration works with software projects too. Iterative development is often seen as a way of controlling costs and mitigating risks, but it's also a great way to capture and verify requirements.

Agile methodologies do this to great effect. An agile project starts off in a certain direction and gives itself a set, short amount of time to deliver some quanta of business value. At the end of this iteration, the team delivers something of business value to the user. This delivery serves many purposes: it helps verify that the developers are delivering the right thing, it potentially gives the users something they can start to use, and it acts as a basis for discussion for future development. Having looked at the iteration, the users are in a much better position to understand what they're getting. This understanding then leads to them refining their requirements.

Powerful feedback takes place as dreams are realized. Anyone can build castles out of clouds, but once the mortar hits the bricks, you soon start to realize what's important and what isn't. We see this effect all the time when delivering software incrementally. When we start off, our users may think they know what they want. But as we start to shape their dreams into reality, and as they start to experiment with the interim deliverables, the users start to see how what they asked for interacts with the rest of the world. They may come to realize that some things were just plain wrong. Other things are acceptable, but would be great if we changed them somewhat. And features that they thought were essential might start shaping up to be pretty marginal when they come to use them. We love this kind of interaction and welcome the changes it forces on our projects. Ongoing user involvement means that the requirements get better understood over time and that the software we deliver ends up being more useful.

This is why we say that requirements (as some absolute, static thing) just don't exist. The benefit of requirements gathering is not the requirements themselves—those will likely change once the project starts. Instead, the benefit is the process we go through while gathering them: the relationships we form with our stakeholders and the understanding we start to develop about the domain. Our July/Aug. 2003 column, "Verbing the Noun," discussed this.

Does that mean that all requirements engineering tools and techniques are unnecessary? Not at all. There are always the constraint-based requirements to track. In fact, in the rapidly changing world of agile development, you could argue that it's even more important that we track these constraints and verify that the changes we make during each iteration's review don't compromise the system's underlying integrity. We just need to recognize that not all requirements are available when we start a project and that the softer requirements that we do capture will likely change over time.

So, while you're reading the rest of this issue about requirements and process, remember one thing. Requirements aren't engineered; they're nurtured.

Dave Thomas and Andy Hunt are partners in The Pragmatic Programmers and authors of the new *The Pragmatic Starter Kit* book series. Contact them via www.PragmaticProgrammer.com.

Classified Advertising

VIRGINIA TECH, Department of Computer Science, Bioinformatics Faculty Positions. The Department of Computer Science at Virginia Tech seeks applications for several tenure-track positions in the Department of Computer Science from individuals desiring to make fundamental contributions to both computer science and the life sciences in bioinformatics, as broadly defined. Special funding from the Commonwealth of Virginia provides competitive salaries and startup funding for tenure-track faculty positions at all ranks. CS faculty in bioinformatics have access to the 2200-processor Terascale Computing Facility recently established by the university, as well as other multiprocessor clusters within the Department. Excellent opportunities for collaborative research exist with researchers in life science departments and at the Virginia Bioinformatics Institute, which is located on campus. Applicants for a senior position must have a significant track record of grant funding. All applicants must have a PhD in Computer Science or an allied area, a demonstrated record of publications in computer science or computational science, and a commitment to addressing significant life science problems. Ability to collaborate with researchers within the Department and in the life sciences is required. A demonstrated record of accomplishments in bioinformatics is preferred. Additional information is available at http://www.cs.vt.edu/FacultySearch. Applicants should send a curriculum vitae, a 1-2 page statement of research goals in both computer science and life science, and at least three letters of reference, as separate PDF files, by email to facultysearch@cs.vt.edu. Review of candidates will begin January 5, 2004 and continue until the positions are filled. Virginia Tech is an equal opportunity employer.

◆◆◆

Programmer Analyst: Responsible for design, analysis, development, maintenance, and testing of logistics, inventory, billing, and financial applications for clients in retail, transportation, manufacturing, and financial industries. Minimum requirements are Bachelor of Science degree in Computer Sciences or Engineering. Must be able to demonstrate proficiency in using the following applications: COBOL/COBOL II, VS COBOL 2, COBOL 74/85/370,JAVA2, VB6, with DB2, IMS, CICS on MVS, JCL, TSO, ISPF/PDF, VSAM, OS390, ES9000, SDF II, DCLGEN, SPUFI, IDCAMS, SUPERC, FILEAID, BOOL AND BABBAGE, OMEGAMON, SPUFI, DCLGEN, IDCAMS, BMC, PANVALET, QUERYMAN, PRINCETON SOFTWARE TOOLS & IBM UTILITIES, EXPEDITOR, ANVIL, REVOLVE. 80% Travel. Qualified applicants may apply by sending application letter and resume to Peak Performance, 500 Waterman Avenue #206, East Providence, RI 02914.

Product

A software company- Pointer-i is offering its product – ELENET. ELENET is an advanced Simulator of the Power Distribution Network. It uses Unix/Linux-ORACLE RDBMS design, modeling and tuning, and is fully Internet enabled. Objective: •Reduce overall loss of power delivery from 2 to 4 per cent. •Estimate the loss from wire heating, unbalanced distribution, reactance compensation, voltage and frequency deviation. •Analyze the estimates comparing actual and optimal statuses of the network. Features: •Graphically portrays and reconfigures the network to facilitate operation, maintenance and planning (much easier to use than AUTOCAD). •Supports versioning of the network layouts to provide an engineer with easy access to past and future generations of the network layout. •Informs the engineer about any operation time-related event (switching, tagging, contingencies, tapping, etc.) •Applies any sequence of the network events to any net generation selected; i.e. any past, present or future network configuration may be simulated under any weather conditions. (This is a unique feature of ELENET). •Expresses overall loss as a function of network configuration and nodal loads in individual estimates of the various loss factors. •Allows graphical entry of x-y dependencies like impact cost functions, typical customer load profiles, weather changes, etc., which is easier and more operable than table entry. •Resembles a multi-channel quasi-real-time metering instrument in providing fast and accurate load flow calculation. •Compares the results of manually set configuration to the optimal configuration to achieve the least amount of overall loss. Methodology: Power flow analysis is based on accurate load flow admittance calculation for radial feeders and Gauss-Seidel method for meshed network. Fast heuristic approach of feeder reconfiguration helps manage the distribution network. Determining voltage regulation settings based on sophisticated research of customer usage patterns. We can implement your network inventory on our system, or build an automatic interface. Please contact us at: 416) 721 7893, 416) 894 3651 ask for Sasha or Yuly. Via e-mail: sashatca@yahoo.com.

SUBMISSION DETAILS: Rates are $110.00 per column inch ($300 minimum). Eight lines per column inch and average five typeset words per line. Send copy at least one month prior to publication date to: Marian Anderson, *IEEE Software*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314; (714) 821-8380; fax (714) 821-4010. Email: manderson@computer.org.