Editor: Suzanne Robertson ■ The Atlantic Systems Guild ■ suzanne@systemsguild.com

System Phenotypes

Alan M. Davis

Is it a requirement or a solution? Is it at the right level of detail? Al Davis has come up with an elegant way of addressing these questions by using a metaphor drawn from the genetics field. Try applying the idea to your project and write and tell us the results. *—Suzanne Robertson*

Throughout my years working in the requirements world, people's inability to comprehend that requirements describe a system's external view has surprised me repeatedly. Perhaps this is because many years ago, somebody defined a requirement as a statement that "specifies what a system is to do without specifying how it does it." Perhaps an entire generation of people now thinks a way to differentiate between a "what" and a "how" really exists. I certainly do not. My how is your what. My what is your how.

In genetics, scientists realized long ago that an organism's external view might hide its internal characteristics. Geneticists use the term *phenotype* to represent an organism's externally observable characteristics. They use *genotype* to represent characteristics that are hidden in an organism's genetic code. So, when we say that a fruit fly has normal wings, everybody understands that we're saying the fruit fly's phenotype includes normal wings. Similarly, we're talking about genotype when we say a fruit fly carries a recessive gene for short wings. If you place the fly under a microscope with successively higher-power lenses, you're simply examining finer details of its phenotype.

Requirements as phenotypes

Stating requirements is the same as defining the phenotype of the system we desire. Like the fly-and-microscope scenario, you can state requirements at increasing levels of detail from the most abstract, or *features*, to the very detailed:

- The system shall allow hotel residents to make long-distance phone calls.
- When the system is generating a dial tone and the user dials a "9," the system shall generate a distinctive dial tone.
- When the system generates a dial tone, it shall be a tone of *x* MHz, plus or minus *y* MHz.

Whenever I present examples such as these, people complain that one example is too vague or another is too detailed. The degree of detail has little to do with whether the example is a valid requirement, just as the degree of specificity has nothing to do with whether something validly describes an organism's phenotype. You base a requirement's correct detail level on the degree of risk tolerance and customer demand. If a customer would accept all interpretations of a vaguely specified requirement, that lack of detail in the requirement is satisfactory. If even one of the interpretations would make the customer unhappy, however, the requirement needs more detail. It's as simple as that.

This worries many customers, developers, and consultants. Product customers regularly ask me, "Isn't this requirement too vague?" and I answer, "Is it?" Because if they think it's too vague, it is. Development personnel often ask me, "Isn't this requirement that my client gave me too detailed? Doesn't it suppress our creativity?" I respond, "Would you rather have your creativity suppressed or have a dissatisfied customer?"

So, a requirement is a system's externally observable characteristic, or phenotype. It's up to system designers to craft the optimal genotype (architecture, design, code, and so on) to realize this phenotype. Some might argue that an overly detailed phenotype unduly restricts the genotype designers. In reality, every requirement limits the design team's available choices. As soon as you record the requirement that

> *The system shall allow residents to move up and down floors through a vertical shaft in the building.*

you eliminate a telephone system from consideration. As soon as you record the requirement that

> *The system shall display the error message "The zip code entered is incorrect. It must be exactly 5 digits, or 5 digits followed by a dash followed by 4 digits."*

you eliminate all solutions that don't generate that exact error message.

Implications for requirements activities

Once you understand that a requirement essentially describes externally observable characteristics (at any level of detail), requirements engineering activities become better defined. For example, *elicitation* determines the problems being experienced or the opportunities afforded customers and users. Elicitation also ascertains external system behaviors that could address these problems or leverage opportunities. *Triage* (see "The Art of Requirements Triage," *Computer*, Mar. 2003) determines which problems, opportunities, and external behaviors you can address when lacking sufficient resources. And *requirements specification* documents the desired external behaviors of the system to be constructed or procured. Notice how often the word *external* appears in these three descriptions.

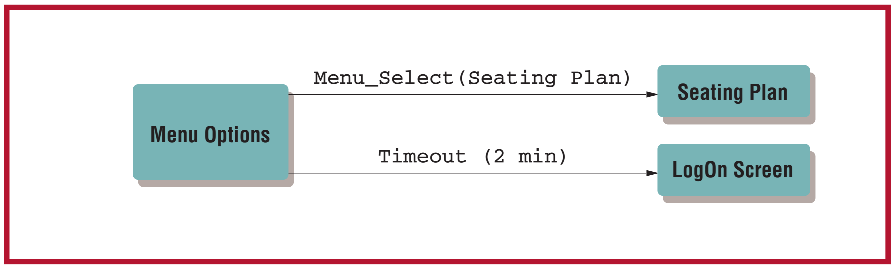

Figure 1. A statechart that captures an elicitation discussion.

Requirements elicitation

During elicitation, the focus must remain on problems and opportunities (which are external to the system) and the solution system's external behaviors. The oft-quoted advice of some expert analysts to keep the discussion on the problems and opportunities and avoid discussing the solution simply cannot work. Humans naturally discuss their problems and opportunities in terms of solutions. Stating "I am hungry" doesn't differ much from stating "I wish I had food." Stating "I feel vulnerable" doesn't differ much from stating "I wish the system would tell me which incoming missiles are the most threatening." Because one of the most important goals of requirements activity is to define a system that pleases the customers, we should spend more time listening to customers than trying to limit their vocabulary. If customers can visualize (and describe) aspects of the solution system that would make them happy, that's good news. Our role as analysts is to understand the customer, not preserve the designers' ideal of creating a system they think is somehow optimal.

Our role as analysts is to understand the customer, not preserve the designers' ideal of creating a system they think is somehow optimal.

During elicitation, analysts often use modeling notations to add structure and understanding to the problem or its solution. This is, of course, beneficial. However, analysts should select notations from among those that are relatively easy for the customer to understand. Furthermore, they should draw using the modeling notation but should speak exclusively in terms customers understand. Let's say, for example, that the application lends itself to modeling using a statechart (see *Modeling Reactive Systems with Statecharts,* David Harel and Michal Politi, McGraw-Hill, 1998). The analyst never needs to use the term *statechart*. Instead, the analyst could say "As I understand it, when the user selects the Seating Plan menu item, you want the system to display a seating plan?" Or, "What I think I hear you saying is that if the user does nothing for two minutes, you want the system to return them to the original logon screen?" While the analyst is saying this, he or she draws it on the whiteboard or computer screen in the form of a statechart (see Figure 1).

You don't need to teach the customer about a statechart. The customer learns about statecharts in situ. Although this is a nice side effect, the spoken dialogue is all in terms of the problem, the opportunity, and the solution system's desired external behaviors. Contrast this with an analyst who draws the diagram just mentioned while saying, "Once in the state named Menu Options, two transitions are possible. The first transition is triggered by the user selecting Seating Plan from among the menu options. In this case, the system transitions to the state Seating Plan. The second transition occurs as the result of a two-minute time-out. If that time-out occurs, the system transitions to the state Log-On Screen." In the first case, the customer is completely on board and might have even learned something. The second case alienates the customer.

You must state all candidate requirements in terms of externally observable phenomena. For example, if a designer wants to make an algorithm faster, that requirement should be stated as, "The system performance shall be improved so that response time (or throughput or capacity) decreases (or increases) from *x* to *y*."

At the end of elicitation, you've created a list of problems and opportunities as you understand them and a list of possible abstract phenotypes (or features) you want the system to possess to address those problems and opportunities.

Requirements triage

Similarly, during triage, the focus must remain on problems, opportunities, and phenotypes. Everything you discuss should be in terms of the requirements gathered during elicitation, plus available resources (for example, people, money, and equipment) and desired delivery dates.

Modeling notations are rarely used during triage, but trade-offs between the benefits of multiple, competing (for the same resources) requirements are the norm. As long as everything you discuss is in terms of benefits to the stakeholders—that is, observable from an external perspective—comparisons are possible. The questions become something like, "Would I rather have the system (a) be 20 percent faster and delivered on time, (b) perform some specific new feature and be delivered on time, or (c) be 20 percent faster, perform that specific new feature, cost $250,000 more, and be delivered two months late?"

At the end of triage, you've divided the list of problems and opportunities into subsets to isolate those that you will address and have done the same to the list of abstract phenotypes to isolate those the system will possess, given available resources.

Requirements specification

Requirements specification must focus exclusively on external behaviors. Remember that requirements should contain enough detail to ensure that the system satisfies the customers. No well-defined line exists between abstract and detailed, but a distinct line exists between externally observable and internal. In genetics, no well-defined line exists between gross and detailed pheno-types, but a distinct line exists between phenotype and genotype.

At the end of requirements specification, you've defined the detailed phenotype of the system to be constructed or otherwise procured.

The parallels between phenotypes in genetics and requirements in system development are many:

- An organism's phenotype and a system's requirement describe externally observable characteristics.
- An organism's phenotype and a system's requirement do not uniquely define a genotype or a design but, in both cases, they limit the possible genotypes or designs.
- Until geneticists learned the difference between the genotype and phenotype, they were confused by how two normal-winged fruit flies could breed and produce a short-winged fruit fly. Until system developers learn the difference between requirements and design, they will continue to have emotional arguments and mass confusion.

Although many parallels exist, some differences do also. The chief difference is that in genetics, the phenotype is the external manifestation of the genotype (the genotype comes first). With system development, the designer's goal is to construct a system's genotype so that it exhibits the behavior defined in its phenotype (the phenotype comes first).

Understanding the role phenotypes play in genetics might assist system developers in understanding the role requirements play in system development.

Alan M. Davis is a professor of information systems at the University of Colorado at Colorado Springs and author of more than 100 papers and two books. He has more than 20 years of experience in industry. Contact him at adavis@uccs.edu.

SOFTWARE ENGINEERING GLOSSARY

Software configuration management domain

(cont'd from p. 53)

software configuration management (SCM): A discipline applying technical and administrative direction and surveillance to (a) identify and document the functional and physical characteristics of a software configuration item, (b) control changes to those characteristics, (c) record and report change processing and implementation status, and (d) verify compliance with specified requirements. See also *software configuration auditing, software configuration control, software configuration identification, software configuration status accounting, software release management.*

—Continued on p. 93