

Design Traceability

Jeremy Dick

Do you have trouble managing requirements changes in your projects? You could benefit from making better use of design traceability.

Systems engineers designing complex products such as satellites, aircraft, and weapons systems are increasingly attuned to the value of tracing requirements through layers of design. The key driver is change management. In such development environments, rapidly assessing the cost of change is vital to project success.

Practically every system includes software, and software usually harbors the lion's share of complexity. So, traceability through software design layers is equally important.



What is traceability?

Traceability is about documenting the relationships between layers of information—for instance, between system requirements and software design. Broadly speaking, this has two benefits:

- *Increased understanding of design.* Many important questions about a project can only be answered by understanding the relationships between design layers. These questions range from “How does the system meet customer requirements?” to “What is this component’s role?” to “What are the system requirements associated with this test step?” Documenting these relationships engenders greater reflection and subjects your thinking to peer review.

- *Semiautomated impact analysis.* Appropriate tool support for representing traceability relationships can make automated analysis of those relationships possible. When it comes to assessing the potential impact of change, your investment in documenting the relationships will pay off by letting you calculate and present graphs of interdependent design artifacts at the push of a button.

Explicit traceability

Figure 1 is an example of explicit traceability from a recent project. It shows a single customer requirement and the four functional requirements designed to satisfy it. The links are purely for traceability, documenting the satisfaction relationship.

Many software development tools manage design relationships—for instance, between modeling elements (such as classes) and source code, or between tasks and source code files. You can potentially treat all such associations as traceability relationships and use them in impact analysis. Such relationships are considered part of *implicit tracing* because traceability isn't their primary function.

Traceability rationale

In addition to linking artifacts, capturing the rationale for relationships can help you understand and analyze those relationships. Consider, for example, figure 1 with rationale added. Since the relationship is one of satisfaction, we could call the rationale a *satisfaction argument*. In this case, the argument applies to the set of links associated with the requirement, as shown in figure 2.

We can apply the approach—called *rich traceability*¹—to other relationships, such as requirements validation, through tests (using a validation argument) or by checking a product’s conformance to standards (using a compliance argument).

Some organizations collect a particular layer’s traceability-rationale set and publish it as a strategy, design, or compliance document.

Reviewing traceability

Most review processes focus on the documents’ or other artifacts’ content and neglect the relationships between artifacts. It’s just as important to review the tracing between layers as it is to review the layers themselves.

Again, the right tool support can greatly assist you in the review process. Rather than struggling with paper documents and a traceability matrix, a tool can do all the donkey work by presenting in electronic form just the information necessary to review the relationship between the documents.

We typically conduct such reviews requirement by requirement. Suppose, for instance, that we’re reviewing a software design against customer requirements. We examine each customer requirement in turn and the design artifacts that trace to it. Then we consider two aspects:

- *Sufficiency*. Is the set of design artifacts sufficient to satisfy the customer requirement? This question ensures that the design covers and satisfies the requirements.
- *Necessity*. Is each design artifact necessary to satisfy the customer requirement? This question protects against overengineering the design.

Collecting the rationale for tracing makes reviews even more effective. Systematically applying such a review can give you greater confidence that requirements will be met.

Change process

Using traceability, you can implement change management in the following stages:

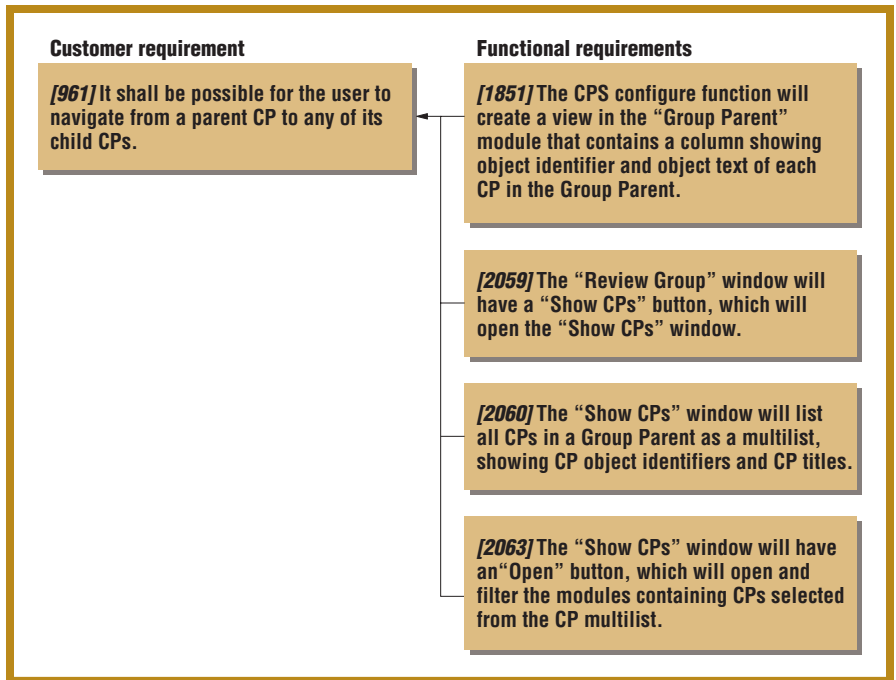


Figure 1. An example of elementary traceability.

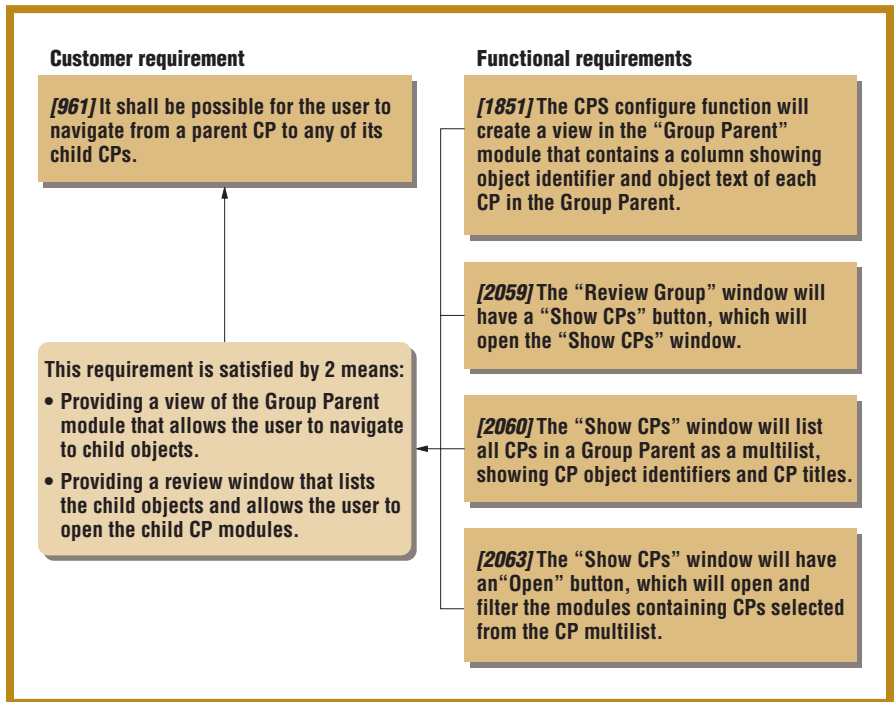


Figure 2. Rich traceability applied to the example in figure 1.

1. *Identify immediate impact*. Determine which artifacts the change affects most directly (for example, which customer requirement has changed or which test case has failed).
2. *Calculate the potential impact tree*. Using an appropriate tool, process the traceability relationships to construct a complete tree of related artifacts up and down the design layers.
3. *Prune and elaborate the impact tree*.

Just because artifacts are linked doesn't mean that a change will propagate. Engineering judgment is necessary for determining which impact-tree branches can be pruned or where you must add new branches because new artifacts are required. Traceability rationale can help you determine the precise nature of change propagation.

4. *Define change.* Traverse the impact tree, working out the precise details of the changes at each point. A configuration management tool can help you do this.

5. *Apply change.* When the changes are ready, apply them to the system in all affected layers.

At each stage, you'll gather more precise information about the nature of the change, including cost. A go/no-go decision point can follow each stage.

Whatever development scale I engage in, I systematically apply information traceability. It's a vehicle for thinking about the way the software meets its requirements; it captures design rationale to help others

understand and review; and it gives me far greater confidence in managing future changes. 

Reference

1. E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, 2nd ed., Springer, 2004.

Jeremy Dick is a principal analyst at Telelogic UK. Contact him at jeremy.dick@telelogic.com.

DESIGN

Continued from p. 13

are essential when evolving the design. Indeed, many view testability as a vital design property, particularly with older systems ("Before Clarity"), leading to design architectures that make the systems more testable ("The Test Bus Imperative").

Designers are people too

Design involves people, so in addition to considering how design fits into a process, you also have to think about how it fits with an organization's people. A common question is, What's the difference between architecture and design?—which raises the question of what an architect's role is ("Who Needs an Architect?"). People often view architects as holding a separate, directing role. However, I strongly believe that technical leaders should work closely with the developers on a team, a principle that also applies to enterprise-wide architects ("Enterprise Architects Join the Team").


This issue of design leadership goes further. Architecture can be about the technical software structure as well as about how the software faces its users—leading to questions that many technical architects don't consider as frequently as they should ("The Difference between Marketecture and Tarchitecture").

Representing design

For a large part of my career, people have talked about representing design in terms of notations, particularly graphical notations that try to tell you important things about a program's structure. As someone who has written books on one of these, I understand both the capabilities and limitations of graphical notations. A common problem with using these notations is that

people use them to represent different kinds of perspectives—even for a single system. Think of three primary purposes for these models: conceptual, specification, and implementation ("Modeling with a Sense of Purpose"). I've come to the conclusion that models are useful for certain tasks, such as class structure or visualizing dependencies ("Reducing Coupling"). However, I don't see them as absorbing the future of software development ("MDA: Revenge of the Modelers or UML Utopia?").

Few of these design principles and discussions are new. It's long been known that you should avoid premature optimization ("Yet Another Optimization Article"), yet constantly we see people doing just that. This is why I spend so much energy simply trying to find good design techniques that have worked well in the past and trying to explain them to others so they'll use them in the future ("Patterns").

So there it is—five years of writing compressed into a single column. I hope that my various authors and I have given you a few useful ideas along the way, and I'm pretty certain that Rebecca will find a lot of good material and be an excellent steward of this column. I will, of course, continue to write on my Web site (<http://martinfowler.com>), and I hope to continue finding useful ideas to write about. 

Submit to RE'06!



14th IEEE International Requirements Engineering Conference

Minneapolis/St. Paul, MN, USA
September 11-15, 2006

<http://www.re06.org>

Details and submission instructions are provided on the conference web pages.

Submission dates

Paper abstracts due	Feb 6, 2006
Papers due (all categories)	Feb 13, 2006
Tutorial /workshop /panel proposals	Mar 6, 2006
Doctoral symposium submissions	May 2, 2006
Posters and research demos	May 2, 2006

General Chair

Robyn Lutz
Iowa State University and
Jet Propulsion Lab, USA

Program Chair

Martin Glinz
University of Zurich
Switzerland