Requirements Tracery

Olly Gotel, Stephen Morris

WHAT DO YOU see when you look at Figure 1? Our guess is that you see a spectacular rose window, one common to many medieval cathedrals. You know that the whole stained glass window is comprised of many pieces of colored glass, each grouped within much smaller constructs often called leaded lights. Daylight passes through each of these lights to form its own small pattern, sometimes figurative, sometimes abstract. What is seldom immediately appreciated is the tracery of stone that holds each of these sources of light, sub-elements of the whole window design, in place and in a particular position. It’s this supporting tracery that facilitates the complex overall design and imagery. The tracery is an integral part of the complete window and is perforce the first part of the window that architects design and put in place.

Requirements resemble these individual glass lights. While they might have some significance in isolation, it’s through their interrelated combinations of meaning that we gain a comprehensive picture of what a software system should do. Not only do we pay little attention to crafting any essential supporting structure in software engineering, we don’t even bother to name it.

Tracery and Traceability

Tracery is a standard architectural term describing the stonework structure that supports the glass lights in a Gothic window.¹ Used metaphorically, the term is associated with something delicate, difficult to construct, and easily broken. Requirements fit this description quite nicely, as they can be challenging to determine, open to multiple interpretations, and subject to changes once specified. Yet, requirements are often woven together without thought to any supporting tracery. This omission is particularly relevant in the context of software engineering because these requirements should inform the development of all the consequent artifacts that will eventually realize the requirements in a working system. Therefore, to provide for ongoing support as requirements become implemented and evolve, the associated trac-

FIGURE 1. Looking up in the south transept of the Cathédrale Notre-Dame de Laon, France. The tracery of stone holds together individual leaded lights of colored glass to form the overall design of a Gothic rose window. (Photo courtesy of John Ferro Sims.)
ery must interweave a myriad of additional artifact types.

Traceability, a term common in software engineering, isn’t tracery. Traceability expresses the potential to create and follow links between pairs of development artifacts to support analytical or development tasks. While a traceability information model or some similar specification of permissible links between artifact types can pre-define the intended types of trace to establish and use, the overall tracery obtained in practice is rarely designed or scrutinized. The sum total of the actual traces created within a project provides the scaffolding for supporting many diverse engineering activities. They deserve far more attention.

Beyond the Matrix
A widely used method to examine traces in practice is to construct a traceability matrix that shows how artifacts of one type link to artifacts of another type, with the semantics of this link often qualified in some way. For instance, the matrix in Figure 2a reveals which requirements depend upon which others, while the matrix in Figure 2b reveals which requirements are satisfied by which elements of the design. While they play a valuable role in software engineering, such matrices only provide for very overt and somewhat partial views of tracery.

The view of the tracery in Figure 2a is overt because it’s problematic to recognize the overall requirements structure; it’s like looking at a rose window and seeing little more than stone ribs. The tracery of Figure 2b is partial because such matrices must be placed in series to examine the full trace of any one requirement through all life cycle phases of a software system’s development. Moreover, as the number of artifacts and links increase, individual matrices become unwieldy to create, maintain and use, let alone to aggregate.

The traceability community has long recognized the need for improved visualizations to render traces visible and support traceability-enabled tasks. However, a cursory look at any requirements management tool will show the continued dominance of the matrix, perhaps augmented by a graphic or hierarchical listing of salient dependencies between artifacts. We suggest that a major barrier to advancement with visualization in this area has been the uncertainty surrounding the nature of the hidden tracery structure that we’ve been striving to render visible in new ways. This limitation is compounded by an equally poor understanding of the intended end uses for such visual representations.

Tracery: A Pattern of Traces
When practitioners engineer requirements and their resulting software systems, they’re creating—alongside and within all other important development artifacts—a web of interconnections that we wish to call requirements tracery. Revealing and understanding the pattern of this tracery has proven critical in many other fields where tracing is fundamental (for example, ichnology and epidemiology), while some even explicitly design the pattern (for example, metrology). We’ve written about the practices of other tracing disciplines elsewhere.²

Some kind of semantic building blocks make up this requirements tracery but their exact and full arrangement is completely unknown. It’s also unknown whether common patterns occur in such arrangements, something equivalent to the configurations of stonework in rose windows. Our best guess at the moment is that the pattern is composed of the multiple tracks that individual requirements leave as they become implemented. Some requirements split, some combine, some halt

<table>
<thead>
<tr>
<th></th>
<th>Req ID1</th>
<th>Req ID2</th>
<th>Req ID3</th>
<th>Req ID4</th>
<th>Req ID5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req ID1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Req ID2</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Req ID3</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Req ID4</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Req ID5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th></th>
<th>DE ID1</th>
<th>DE ID2</th>
<th>DE ID3</th>
<th>DE ID 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req ID1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Req ID2</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Req ID3</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Req ID4</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Req ID5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b)

FIGURE 2. Simple example of traceability matrices: (a) shows which requirements depend upon which others—that is, Req ID3 depends upon Req ID1—and (b) shows which design elements satisfy which requirements—that is, Req ID3 is satisfied by DE ID2.
prematurely, and some suddenly appear or reappear for no discernable reason. Our appeal through this column consists of four ideas. First, if we’re going to start paying attention to improving and exploiting the structure that supports our requirements and other development artifacts in software engineering, let’s start by naming this structure. Requirements tracery is our proposal.

Second, let’s begin to render requirements tracery visible and examine its pattern across different types of projects and contexts. As an artistic by-product of these efforts, the traceability community could even curate a gallery of our discipline’s unique “rose windows.”

Third, let’s learn about the impact of various tracery patterns on the volatility and viability of managing requirements and on the support of other important traceability-enabled tasks. Ultimately, let’s use the insight gained to either inform the design of a requisite requirements tracery as an integral part of the software engineering process or help guide the recovery of the traces on demand later.

If our requirements tracery is created well, it will be an engineered structure, purposefully crafted and exposable as needed. Its presence in end products will be as imperceptible to end users as the stonework to viewers of rose windows—in the process, perhaps our software systems will acquire some of the windows’ splendor and longevity.

References

OLLY GOTEI is an independent researcher working in New York City. Contact her at olly@gotel.net.

STEPHEN MORRIS is an honorary senior visiting research fellow in the Department of Computing at City University London. Contact him at sjm@soi.city.ac.uk.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.

RUNNING IN CIRCLES LOOKING FOR A GREAT COMPUTER JOB OR HIRE?

Make the Connection - IEEE Computer Society Jobs is the best niche employment source for computer science and engineering jobs, with hundreds of jobs viewed by thousands of the finest scientists each month - in Computer magazine and/or online!

IEEE® computer society | JOBS

http://www.computer.org/jobs

IEEE Computer Society Jobs is part of the Physics Today Career Network, a niche job board network for the physical sciences and engineering disciplines. Jobs and resumes are shared with four partner job boards - Physics Today Jobs and the American Association of Physics Teachers (AAPT), American Physical Society (APS), and AVS: Science and Technology of Materials, Interfaces and Processing Career Centers.

> Software Engineer
> Member of Technical Staff
> Computer Scientist
> Dean/Professor/Instructor
> Postdoctoral Researcher
> Design Engineer
> Consultant