Requirements Engineering in Automotive Development: Experiences and Challenges

Matthias Weber and Joachim Weisbrod, *DaimlerChrysler Research*

I n the automotive industry—especially in the high-end market—the complexity of electronic components is increasing rapidly. Currently, about a third of all development costs in high-end models go to electric and electronic system development, and the cost continues to grow (see Figure 1). At the same time, many slightly different variations on components are each developed in a series of prototyping phases on different schedules. Consequently, the complexity of specification activities surpasses

Complex automotive systems yield complex requirements specifications and raise many challenges along the way. Using real–world projects as a foundation, the authors describe problems and solutions for requirements engineering in the automotive domain.

what conventional text-processing systems can support in terms of management and tracing functionality.

For the past few years, DaimlerChrysler has addressed these challenges by piloting requirements engineering processes, methods, and tools in various development projects. Our experiences have helped us identify and address various RE challenges, including those related to the software-based automotive systems domain, which we describe briefly below, and those related to domain-independent requirements administration and management. Although the issues we discuss here are not necessarily the current focus of RE research, we think that progress on these issues is needed for a widespread adoption of tool-supported RE in the automotive domain, and many other domains as well.

RE projects

At DaimlerChrysler research, our RE projects cover three primary areas:

- *Entertainment and telematics*. In-car telematics has become a major challenge for automotive development because of the technology's rapid evolution and high visibility and its underlying system's complexity. At Mercedes-Benz Technology Center, about 60 engineers design and specify the telematics system for a single car series.
- *Car interior and passenger comfort*. In this area, the overall system has some 70 interconnected features, each of which has, on average, a 50-page specification.
- *Driver assistance and safety-critical systems*. In both passenger cars and com-

mercial vehicles, we are developing more and more systems that can intervene with the driver's actions (examples include the electronic stability program, the antilock braking system, and drive-by-wire). In such systems, failure can cause severe accidents. Therefore, we have a demanding development process: system specification and system development must be completely comprehensible, reproducible, and continuous.

Project sizes range from small efforts, with two to five developers working on a requirements database that generates a few specification documents, to significantly larger projects. Currently, our biggest project has a requirements database that is approximately 3 Gbytes (and growing) and generates more than 20 automotive component specification documents. The project has about 160 users working on the database, along with 40 users with read-only access.

In our RE work, we've used three requirements management tools: Dynamic Object Oriented Requirements System (Doors; see www.telelogic.com/products/doorsers/doors); RequisitePro (www.rational.com/products/reqpro), and Requirements Traceability and Management (RTM; see www.chipware.com). While all three support basic database management for individual requirements, their attributes, and their history, the tools widely differ with respect to the user interface, tracing and view management, distributed work, security concepts, extensibility mechanisms, and import–export interfaces.

Our RE work has produced several important results. First, we established an RE team within DaimlerChrysler research that supports the company's business units. We then defined a generic RE process to both coordinate the RE team's work and support discussions with the business units. Several business units subsequently adopted the generic process and tailored it to specific customers' needs. Additionally, we

■ Established several DaimlerChrysler-specific requirements management information models
■ Specified guidelines for requirements management tools tailored to DaimlerChrysler processes
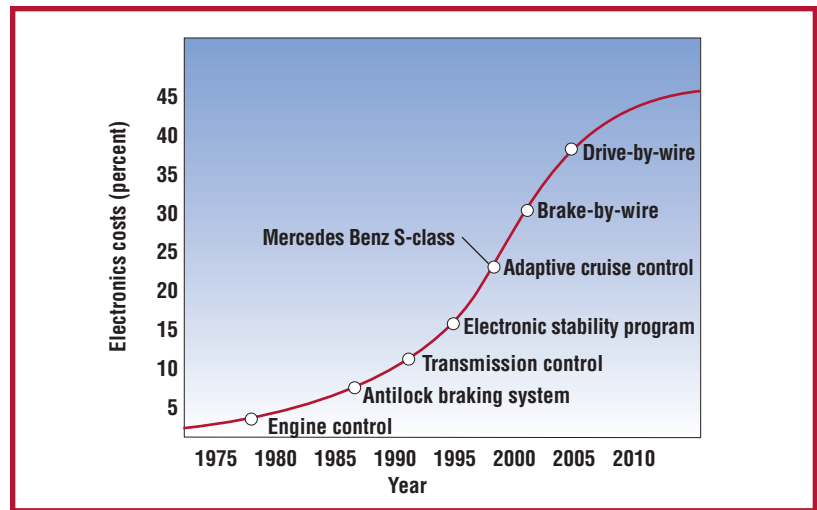■ Evaluated our experiences and worked

Figure 1. Ratio of electronics development to overall development costs.

on linking requirements management tools to other DaimlerChrysler tools
■ Developed research prototypes and case studies of best practices in RE

We also supported many pilot projects introducing the new processes and tools. Doing so let us provide add-ons to requirements management tools as project engineers requested. In particular, we developed 10 specific functional extensions to Doors with about 50 thousand lines of code. One of these extensions lets us specify and generate documents that integrate data distributed in the Doors database. Another supports advanced definition and automated administration of Doors projects, while several others define additional export and import interfaces. Some of these extensions are needed for other requirements management tools and nonautomotive applications; requirements management tools have a ways to go before they'll support sufficiently mature functionality and appropriate automation levels. In the future, we expect many of these extensions to become unnecessary due to tool advancements. We distribute updates of our extensions and their documentation—along with company-specific material on requirements management methodology and process—to all Doors users within the company twice a year.

General issues

As we outline below, some of our problems centered on technology, while others were focused on process. Here we describe experiences that were strongly related to both.

Information models for requirements management

Textual requirements are only part of the game—automotive development is too complex for text alone to manage.

This does not mean, of course, that textual requirements are unimportant. However, tackling and controlling just the textual requirements is clearly insufficient. Moreover, when it comes to complex systems, controlling "just the textual requirements" is nearly impossible. To reflect the characteristics and phases of the specification and development processes, we need additional attributes, such as working state (initial, defined, and released, for example) for the former, and due date (such as A-sample and B-sample) for the latter.

Researchers now openly acknowledge the importance of such attributes, and useful collections exist.[1-6] The trick is to tailor the right set of attributes so that the effort to define and maintain them is balanced by the benefits of better process control and specification reuse. We must also handle the requirements' history. In addition to recording the local history of a single requirement's manipulation, engineers typically need a well-structured global history of how a specific group of requirements was manipulated relative to some previous baseline.

Along with attributes and history, we also must deal with a specification's illustrations, tables, explanations about design rationale or design decisions, test information, parameters, and accompanying background information (such as excerpts from standards). This additional information often makes up more than half of the overall specification.

Often, the complexity of these different objects, their attributes, and the relations between them quickly confuses engineers. To remedy this situation, we formally defined all this information using a metadata model. A requirements management information model (RMI) is central to requirements management projects: It forms the basis of discussions with engineers about their needs and how they should systematically manage their specifications. Developing an RMI with engineers has become our primary method for successfully introducing requirements management into projects.[7] To ensure consistency and reusability, we introduced a company-wide, modular RMI that new projects can adapt and tailor.

Requirements documents

Engineers live in a document-oriented world, and requirements management tools must maintain its look and feel.

System developers typically focus on document-centered RE. This is unsurprising because documents are the traditional internal and external interface to suppliers. In fact, the entire development process (and especially a formal contract) is based on documents and their exchange. Introducing database-based requirements management tools can thus be most successful if they retain a document's look and feel.

Tool support should retain the basic functionality of a standard text-processing user interface while adding management and tracing functionality. Tools that deviate from basic aspects of standard text processing are quickly discarded in real-life projects. This does not mean that users should know nothing about the underlying database; on the contrary, when working on requirements filtering or tracing, for example, they must have a profound knowledge of the underlying logical database structures.

Tools must also contend with nontextual document information. For example, recording the history of pictures is tricky because users typically record a picture's change history only on the picture level. Picture size also causes serious problems, and tools must support compressed formats (which they typically don't).

Also, because users will always have documents that contain information outside the requirements management tool database, you should integrate requirements management tools with document management systems. You can only achieve this at the granularity of entire requirements management databases.

You should also make data entry as comfortable as possible. For example, you should offer graphical support for editing specialized input forms, as in standard database systems. Finally, users will never accept requirements management tools unless they provide flexible, easy-to-configure, and efficient document-generation facilities that satisfy given standards and combine and filter material of all kinds. Engineers must be able to quickly produce both their main

specification documents and the various reports that management requests.
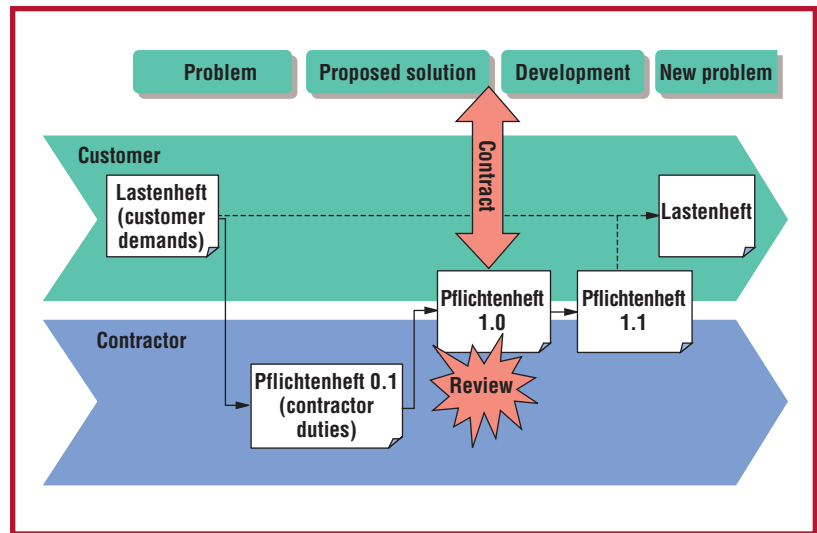
Problem and solution space

There is no clear boundary between manufacturer requirements specifications and supplier system specifications.

In German, there is a clear distinction between *Lastenheft* (literally, "demand booklet") and *Pflichtenheft* ("duty booklet"). As Figure 2 shows, the customers provide the first document, which defines what they want (the customer demands), and the contractors provide the second document, which explains how they'll build the system (the contractor duties).

At first glance, the English counterparts to these documents are the user requirement specification and the system requirement specification. The user requirements document supposedly tells what the problem is, while the system requirements document should describe the solution. In the real world, however, things are not that easy because customer demands are not limited to the problem space. In the automotive domain particularly, there are several good reasons for customer descriptions to go beyond the problem space:

■ A system is typically divided into subsystems and components that different suppliers provide. DaimlerChrysler must ensure that different suppliers' components will work together to specify parts of an overall solution.
■ When we try to separate software from hardware things get even harder. We must specify a single component as two interactive "problems"—a hardware problem and a software problem.
■ In many situations, DaimlerChrysler's first-to-market strategy requires us to define certain specification document features in as much detail as possible, which may be a concrete algorithm. Often, DaimlerChrysler must set the future standard by specifying solutions instead of problems.

In some situations, it can even be challenging for the RE expert and the domain engineer to decide whether a given requirement should be part of the customer-demands document.

Figure 2. The interaction and distinction between customer demands (*Lastenheft*) and contractor duties (*Pflichtenheft*) in automotive development.

Redundancy

Redundancy causes good engineers to suffer, and the resulting systems will probably suffer, too.

In large projects, such as automotive electrical system development, engineers simultaneously create several hundred lengthy specifications and related documents (of up to several hundred pages each) under tight deadlines. It is not surprising that these documents typically contain redundant information. Furthermore, dependencies among such documents are complex and remain partially opaque due to both the lack of time and the number of documents and people involved. The disadvantages here are obvious: This form of redundancy leads to inconsistencies and double work. In the worst case, specification documents are derived from invalid premises.

This is a vexing problem because document structures have often evolved and gained acceptance over many years. Moreover, the document structure typically mirrors organizational structures and even (in our case) the entire automotive business structure. Changing document structure to remove redundancies is therefore difficult, and there is often a negative trade-off between effort and result.

In our experience, getting an overview of project redundancies is a major step. Doing so makes engineers aware of the parts of their documents that are used or elaborated somewhere else. They can then decide to synchronize if necessary.

Content presentation

Content does not cause suffering for good engineers, but content presentation can create problems.

> **Well-structured requirements and design decisions at several layers of abstraction are crucial for understanding a detailed specification document.**

In the automotive domain, we typically build systems in increments, rather than from scratch: A new car series inherits most functionality from existing ones, with various adaptations, extensions, or innovations. The new windshield wiper is basically just another (and even better!) windshield wiper.

Consequently, at this level, there is no formal elicitation phase because developing a new component version typically entails few related activities. We've found that good engineers are premier league experts in their specialized domains and have deep insights into the subtleties and pitfalls of requirements content. For them, content and domain knowledge is not a significant problem.

However, content presentation and structuring offers challenges. Often, engineers describe specifications in an unstructured way, which increases the communication effort. A well-defined RMI implemented by a requirements management tool, however, can help improve content structure and presentation. For example, it can push engineers to break up long specification paragraphs into atomic requirements.

Reuse is another point of improvement. Today, most specification reuse is ad hoc and implicit, which affects efficiency and quality. The main problem with this approach, however, is that a good specification depends completely on an engineer's advanced expertise in the domain. As an alternative, we propose systematically recycling existing specifications as an explicit step in the specification process.[8-10]

Finally, when you introduce a requirements management database, you must migrate existing documents into the system so that they're available for reuse in future projects. Although requirements management tools support some automatic capturing, in our experience, the migration process still requires significant manual work.

Process issues

Our process-related experiences included challenges with abstract specifications, nonfunctional requirements, change, user-adaptable views, and specification reviews.

Abstract specification levels

Detailed specification typically covers only the leaves of the system decomposition tree.

Engineers cannot develop a complex system—such as an up-to-date telematics unit—solely by talking about and dealing with the system requirements that make up the low-level, detailed component specification. On the contrary, developing complex systems from the top down in several layers of granularity is inevitable.

When it comes to RE, this observation still holds. Unfortunately, today systematic processing and documentation of higher-level requirements and design decisions are insufficient. However, if engineers don't document these high-level requirements and rationales, they build their detailed requirements on sand: Some will be challenged regularly because they're hard to justify (even though they might be of central importance); others are completely superfluous but still exist (like the requirement someone derived two years ago from a subsequently suspended legal regulation).

Last, but not least, well-structured requirements and design decisions at several layers of abstraction are crucial for understanding a detailed specification document. There are many description techniques, including goal trees, feature lists, use cases, message sequence charts, and state models. The challenge is defining a domain- or organization-specific model. To this end, our own research has mainly focused on use cases.[11-13]

Nonfunctional requirements, acceptance criteria, and test cases

Engineers fail to manage nonfunctional requirements, acceptance criteria, and test cases.

Every engineer at DaimlerChrysler understands the importance of acceptance criteria, test cases, and nonfunctional requirements. Unfortunately, even in projects with the appropriate resources, actual specification documents rarely meet the corresponding claims because it's hard to come up with reasonable acceptance criteria and appropriate test cases for functional requirements. When it comes to nonfunctional requirements, the situation is even more difficult.

From our experience, writing good acceptance criteria is simply a matter of practice and personal experience. Therefore, the best way to improve acceptance criteria is to

provide concrete examples for functional requirements and have engineers exploit and reuse these examples. Basically, this is just another argument to foster systematic requirements recycling.

As for nonfunctional requirements, postulating a quality such as maintainability is hard to substantiate. Generally, corresponding acceptance criteria and test cases are either nonexistent or too academic—"90 percent of all maintenance jobs should take less than five minutes," for example—so these requirements are practically meaningless. Whenever possible, our experience indicates that you should stepwise refine nonfunctional requirements until you can implement them as a set of functional requirements.

Change

Change and discussions about change are part of daily project life—and there's no way to change that.

All kinds of changes occur during projects, including late and totally unexpected ones. You might criticize the amount of change, but we think that's naive. As long as changes are justified and technically feasible, they will occur. Consequently, you must ensure that you can reasonably handle late and unexpected changes.

Among the possible alterations in a project are changes in system, application, and component requirements; changes in the RMI, the schedule, responsibilities, and suppliers; and even changes in the processes and tool environment. A major underlying reason for the frequency of change is the growing complexity, parallelization, and time restrictions on development projects. These factors force developers to introduce more and more assumptions about the system early in development. They then often challenge or rework these assumptions as the system matures.

We were able to limit RMI changes to essential ones by establishing standard RMIs across projects and minimizing nonessential adaptations. We also found that change proposals occur far more often than actual project changes (and they typically increase late in development). As technical experts, engineers must quickly formulate opinions about change proposals. If RE is based on an appropriate RMI, engineers can evaluate
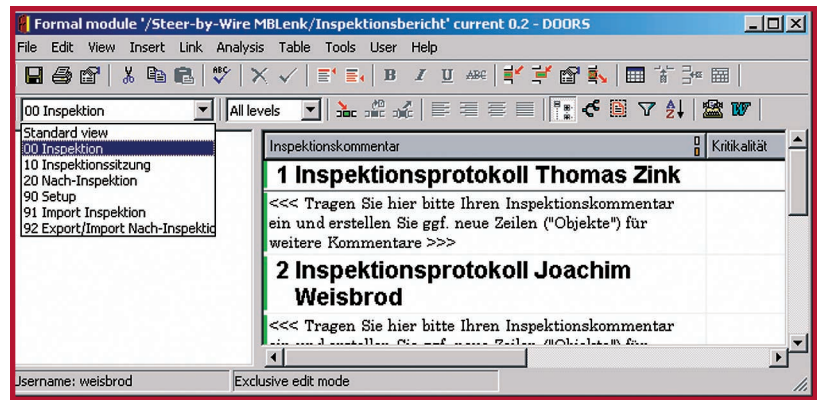

Figure 3. Different workflow phases for the inspection process.

a proposed change's potential effects based on documentation of relations among requirements and corresponding attributes.

Unfortunately, at upper levels of system decomposition, the lack of appropriate specifications hinders the analysis and propagation of local changes introduced late in the development process. This in turn reduces specification reusability in subsequent projects.

User-adaptable views

User-adaptable views are a powerful instrument for guiding and managing the specification process.

In a complex domain, RE is a complicated, continuously modified process with many different activities. On most projects, people play various roles with complex dependencies, and different people read, modify, review, accept, or release different aspects of the specification at different times. By supplying both user- and situation-specific database views, the requirements management tool lets you coordinate and support the RE process at a fundamental level. In some projects, the requirements management tool's view support also helped with workflow management (see Figure 3).

Automatically computed views are key to getting engineers to adopt a new tool. However, to make this work, the view concept must support two important requirements. First, the tool must offer powerful support for defining individual views: It should support not only attribute selection and filtering, but also automatic information generation and systematic information display, possibly through multiple linking steps. Views should also support user access rights.

Second, tools should offer view management support. Engineers should be able to program complex role- or phase-dependent

views for reuse across projects. Standardized views will thus become part of the RMI. Engineers should also be able to easily adapt views or create new simple views. They might do this, for example, to argue for some point during discussions with management or during review meetings with suppliers.

Many engineers are not used to developing their documents "in public"—and they don't like the idea. Keep this in mind when it comes to user acceptance, and consider letting engineers work in a private part of the database (at least in early task stages).

Reviews

Well-organized specification reviews are essential for a successful manufacturer-supplier relationship.

Informal and formal reviews occur frequently as specifications steadily evolve. These reviews are important to assure both specification quality and subsystem compatibility and sometimes represent the only measure to guarantee overall system compatibility and integrity.

The automotive domain is complex, and engineers are typically responsible for several component versions in different development phases. In such a setting, it's both ambitious and important to establish effective and efficient reviews.

In our experience, effective requirements management practices and appropriate tool support are the key to successful specification reviews. The single source approach ensures that every inspector reviews the same version of the review artifact. Role-specific views and attributes ensure that everyone knows what to do and how to do it in each phase. In review meetings, we document decisions online. We also provide project management with special views to control the review process. Last, but not least, the requirements management tool's history features let us trace every decision and question them at any point.

This kind of appropriate requirements management support substantially improves and facilitates the review process. This was a major selling point when it came to convincing both engineers and project management of the usefulness of systematic requirements management practices and tools.

Technical issues

Our technical experiences included challenges related to traceability, tool integration, and the need for more advanced requirements management tool support.

Traceability

Traceability is a great feature, but the real challenge is deciding which traces to maintain.

Introducing explicit traceability promises big savings,[14, 15] but achieving them requires a careful tradeoff analysis. There are various potential links among objects. And, while linking between requirements and related objects is essential, maintaining links requires effort that should be balanced by traceability's benefits. To this end, our advice is to basically leave these decisions to engineers because they know what they want and what they can accept. Generally speaking, engineers are initially keen to establish and maintain all kinds of traces, but later they fall short on the discipline required to do so. With this observation in mind, we recommend that you keep the number of explicit traces small.

Technically speaking, there are two ways to express object relations:

- Explicit links, represented by link objects
- Implicit links, such as textual references integrated into text (such as, "see SR-123") or conventions (such as, "Chapter 12 in document *XX* corresponds to Chapter 4 in document *YY*")

The RMI houses all decisions about which objects to trace and whether to trace them implicitly or explicitly, and you should precisely define these decisions during project inception. Design guidelines can help you master traceability by defining rules about which links to establish, how to establish them, and when to update them.

As for requirements management tools, their link support is still inadequate. Mainstream tools do not support links as first-class citizens in terms of link histories and attributes or distributed work on links, for example.

Tool integration

Couplings between textual specification and modeling tools are immature and seldom used.

Although most specification activities are still document-based, a growing number of specifications require complex models, such as executable analysis models, system and software design models, and HMI models. This requires engineers to develop a specification using two or more tools: a tool for textual specifications and one or more tools for model-oriented analysis and design.

On the basis of customer demand, tool vendors have developed script-based couplings between requirements management tools and modeling tools. However, we found that most tool couplings—which usually originate in a specific project and were designed and paid for by a specific customer—are insufficiently mature for serious development project use. Several of our projects started using such add-ons but later dropped them, even when doing so would clearly require considerable manual effort or significant process problems.

Among the shortcomings we found were

- *Speed*. In an average project, the number of linked objects can easily grow to several thousand, which results in unacceptably slow coupling speed (when calculating changes, for example).
- *Integrated document generation*. Existing tool couplings don't support this feature, whether the documents are short status reports or lengthy documentation that satisfies a standardized structure. Although a few tools are dedicated to integrated document generation, they exist only in specific vendors' tools suites and are largely useless outside of them.
- *User interface*. Tool couplings usually create redundant editors for managing cross-tool information (typically one for each tool involved).
- *Automation*. In most tools, the automation level is low, there is no active administrative support, and users must initiate synchronization. Also, there is no active support for indicating problems.

To pragmatically deal with these problems, you can extend a model-based specification tool to manage textual specifications (including formatting, attributes, views, history, and baselines). A more methodological approach is to establish design guidelines that keep the text–model interface lean, permitting links only between certain text and model elements, and only then during certain process steps.

In our view, the long-term solution will come from two directions: Technically, XML-based approaches will permit more systematic, efficient, and uniform tool couplings; and methodologically, improvements will come from further standardization of requirements and system engineering languages. In particular, the Unified Modeling Language unification effort should be extended from software engineering to system and control engineering languages to reduce the current zoo of overlapping and proprietary notations. Further, including requirements management concepts in UML might put all specification- and system-architecture-related information in a single model with uniform modeling concepts.

Requirements management tools

Requirements management tools are the number one instrument for leveraging RE practices— which means they still must be improved.

Many of our challenges have been tool-related. Tools are important because they help leverage improved practices and processes. When engineers ask for improved tool support, you have a great opportunity to improve processes and practices on the fly. The obvious and serious drawback, however, is that if they reject a tool due to deficient features or performance, they'll also reject the corresponding process improvements. Requirements management tool support thus represents both an opportunity and a risk in RE process improvement.

With this in mind, we postulate the following additional requirements for adequate requirements management tool support:

- To leverage process improvement and RE practice adoption, requirements management tools should provide basic workflow support, such as powerful filter and view capabilities and sophisticated view management.
- RM tools must be easily adaptable using standard programming languages so that users can quickly include new functionality. Otherwise, they'll drop the tool because tool vendors are generally too slow to react to urgent project needs.

When engineers ask for improved tool support, you have a great opportunity to improve processes and practices on the fly.

About the Authors

Matthias Weber is responsible for requirements management research at the Software Technology Research Lab of DaimlerChrysler. His research interests include requirements engineering and embedded systems development. He has a PhD in computer science from the University of Karlsruhe, Germany, and is a member of Gesellschaft für Informatik (GI). Contact him at DaimlerChrysler AG, Research and Technology, Methods and Tools (RIC/SM), Alt-Moabit 96A, D-10559 Berlin, Germany; Matthias.N.Weber@DaimlerChrysler.com.

Joachim Weisbrod currently manages the software development process for Mercedes-Benz truck development at DaimlerChrysler; he previously managed requirements engineering processes at the company's Software Technology Research Lab. His research interests include requirements engineering and software engineering in general, particularly in the embedded systems domain. He has a PhD in computer science from the University of Karlsruhe, Germany. Contact him at DaimlerChrysler AG, Technical Systems, Mercedes-Benz Truck Development (EL/T), HPC B104, D-70546 Stuttgart, Germany; Joachim.Weisbrod@DaimlerChrysler.com.

■ Users should be able to control tools off-line via appropriate APIs (or at least command line interfaces), so they can quickly construct and adapt tool links.

In addition to these vital requests, the details often lead to tool rejection in everyday work. If, for example, it takes eight hours for the requirements management database to generate text documents, the engineer who must distribute documents regularly is unlikely to give the tool an award.

Given that the experiences we describe are based on real-world development projects in various automotive domains, it's not surprising that we discovered a huge and seemingly disparate variety of issues. Many issues are concerned with requirements management, regardless of their content. Many others are related to tool support for requirements management.

On the other hand, it might be somewhat surprising to note that we found fewer and certainly less prominent problems related to the specification of actual requirements content (that is, requirements elicitation and notation expressiveness), both of which are the focus of considerable work in the RE community. This might be because developing functionality from scratch is the exception rather than the rule in the automotive domain, where engineers must take a product-family development approach. By extending existing functionality in an evolutionary manner, they somewhat blur the academic distinction between problem and solution space. Consequently, they've built up significant human expertise on the actual components and their functionality. These experts do not typically have a pressing problem with specifying what the system should do; their concerns are

in managing the growing complexity of specification activities and artifacts.

Many characteristics of the automotive domain apply to other complex system engineering domains as well, particularly in areas using or considering product-family development. Thus, while not the current focus of RE research, many of the issues we raise here will be crucially important for improving RE in the future. ⑨

References

1. S. Robertson and J. Robertson, *Mastering the Requirements Process*, Addison-Wesley, Boston, 1999.
2. K. Wiegers, *Software Requirements*, Microsoft Press, Redmond, Wash., 1999.
3. I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, New York, 1997.
4. G. Kotonvy and I. Sommerville, *Requirements Engineering*, John Wiley & Sons, New York, 1998.
5. B.L. Kovitz, *Practical Software Requirements*, Manning, Greenwich, UK, 1999.
6. D. Gause and G. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, New York, 1989.
7. G. John et al., "Using a Common Information Model as a Methodological Basis for a Tool-Supported Requirements Management Process," *Proc. 9th Int'l Council on Systems Eng. Symp. (INCOSE)*, A. Fairbairn et al., eds., Ascent Logic Corp., Brighton, UK, 1999, pp 59–67.
8. J.L. Cybulski and K. Reed, "Requirements Classification and Reuse: Crossing Domain Boundaries," *Proc. 6th Int'l Conf. on Software Reuse*, LNCS vol. 1844, Springer-Verlag, Berlin, 2000, pp. 190–210.
9. K. Wiegers, "Requirements When the Field Isn't Green," *Software Testing and Quality Eng.*, vol. 3, no. 3, May–June 2001; www.processimpact.com/articles/reqs_not_green.pdf.
10. W. Lam, J.A. McDermid, and A.J. Vickers, "Ten Steps Towards Systematic Requirements Reuse," *Requirements Eng.*, vol. 2, no. 2, 1997, pp. 102–113.
11. F. Kiedaisch and I. Alexander, "Towards Recyclable Requirements," *Proc. 9th Ann. IEEE Int'l Conf. and Workshop on Eng. of Computer-Based Systems*, IEEE CS Press, Los Alamitos, Calif., 2002, pp. 9–16.
12. A. Cockburn, *Writing Effective Use Cases*, Addison-Wesley, Boston, 2001.
13. D. Kulak and E. Guiney, *Use Cases: Requirements in Context*, ACM Press, New York, 1999.
14. B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability," *Cooperative Requirements Eng. with Scenarios* (Crews), 1999; ftp://sunsite.informatik.rwth-aachen.de/pub/CREWS/CREWS-99-13.pdf.
15. O.C.Z. Gotel, *Contribution Structures for Requirements Traceability*, doctoral thesis, Dept. Computing, Imperial College, London, 1995.