CHAPTER 16

STORY USE AND REUSE IN AUTOMOTIVE SYSTEMS ENGINEERING

Frank Houdek[1] and Thomas Zink[2]
[1] *DaimlerChrysler, Ulm, Germany*
[2] *Nokia Mobile Phones, Ulm, Germany (formerly at DaimlerChrysler)*

THE SPECIFICATION volume needed to develop a premium car is amazing: The set of vehicle features and functions implemented in software now exceeds 10^4 pages. The countless requirements allocated to dozens of subsystems are described in intricate detail. They are influenced not only by market demands but also by technological opportunities and cost constraints.

Consequently, it is a non-trivial task to keep track of all the rationales behind individual requirements, or the interactions of all the functions. If new functions come into play or existing functions have to evolve because of changed constraints, it becomes hard not to get lost in detail.

In this context, we found scenarios to be beneficial in a range of activities supporting different stages of product definition and enhancement. They proved to be a good means of negotiating new product features, to provide meaningful rationales for requirements (without being too detailed), to provide a users' perspective on technological driven product features and to keep an overview in the chaos of detail.

To illustrate these benefits, the chapter gives first a brief overview of the characteristics of automotive software development. Then we sketch the benefits of scenarios for different activities in the specification area by typical examples.

TYPE OF PROJECT

Automotive software development (e.g. embedded systems, usually functionality distributed over various individual controllers).

APPLICABILITY

The approaches described in this chapter may be beneficial for many types of projects. Projects that could gain most from the approaches include:

Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle. Edited by Ian F Alexander and Neil Maiden
© 2004 John Wiley & Sons, Ltd. ISBN: 0-470-86194-0

- mass market products (i.e. no clear customer who really knows what he wants),
- complex systems with many (interacting) features and functions,
- projects whose stakeholders have (very) different backgrounds, and
- long-lasting projects (since rationales for requirements may be lost over time).

POSITION IN THE LIFE CYCLE

Requirements Discovery	**Requirements Validation**	**System Specification**	System Design	Coding, First of Class	Integration & Testing	Operations & Maintenance

ROLES PLAYED BY SCENARIOS

In our context, we used scenarios for

- communicating requirements from customers representatives to developers,
- identifying necessary interactions of available product features to maximise users benefit,
- providing an overview on detail-level specifications, and
- setting the context for individual requirements to increase understanding and reusability.

STRENGTHS

The strength of the chosen approach is its informality. Writing and reading stories is an everyday experience. It is an ideal means of communicating and documenting essential facts, both for non-specialists, and for engineers when they need to understand the 'bigger picture' around the software they are specifying.

WEAKNESSES

Informality has its shortcomings, too. There is no way of ensuring an informal description is complete. We use stories mainly in an 'opportunistic way', that is, where appropriate, but not throughout a project.

INTRODUCTION

Although cars do not look that different from how they looked 30 years ago (see Figure 16.1), they have changed dramatically underneath their steel surfaces. They are no longer just masterpieces of mechanical engineering—instead they have turned into

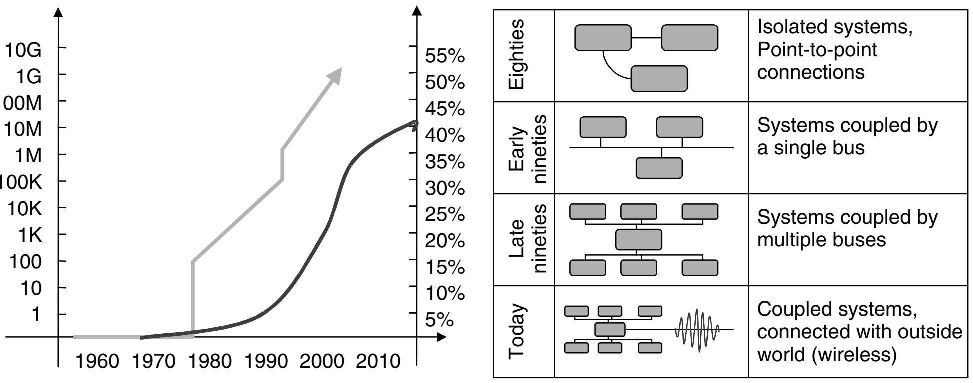
FIGURE 16.1 Some products of DaimlerChrysler (Courtesy of DaimlerChrysler AG)

complex computing devices embodying numerous processors and hundreds of thousands of lines of code. In the automotive domain, software has become the main driver of innovations and new functions. Of all upcoming innovations, 80% will be driven by electric/electronics systems, and of those, 90% are implemented in software (McKinsey, 2002).

Defining these new software functions, however, is a complex activity. As for all systems developed for the mass market, there is no really competent customer available in the sense that you can ask about the required features and functions and he can give precise answers. Instead, one has to triage potential new functions between technical feasibility, (potential) market demand, and cost (Davis, 2002). In this process, we found scenarios to be beneficial for various activities, as they helped us to focus on the essence of new or modified functions.

AUTOMOTIVE SOFTWARE DEVELOPMENT

In this section, we give a brief overview of automotive software development and its special characteristics. As Figure 16.2 illustrates, the volume (a), the contribution to the entire value chain (b) and the complexity (c) of automotive software has increased dramatically over the last three decades (DaimlerChrysler, 2002). Software started its success story in vehicles as a small supplement to classical electronic solutions, and has become a central part of today's complex distributed systems.



(a) code volume: dashed line
(b) contribution to value: solid line
(c) software complexity

FIGURE 16.2 Growing volume, contribution to value chain and complexity of automotive software over time

A recent medium to upper market segment vehicle typically embodies 40 to 100 electrical control units (ECUs, see e.g. Figure 16.3). The amount of software in all of these ECUs together has reached the order of 10^5 to 10^6 lines of code. In the recent Mercedes Benz S-class, for instance, there are more than 50 ECUs embodying more than 600,000 lines of code. The ECUs are connected by three buses with several hundred bus messages (Grimm, 2003). There are gateways between the buses, and functions are heavily interconnected. Via wireless interfaces, subsystems can connect to external systems, for instance, to receive traffic information or to report the car's position in an emergency.

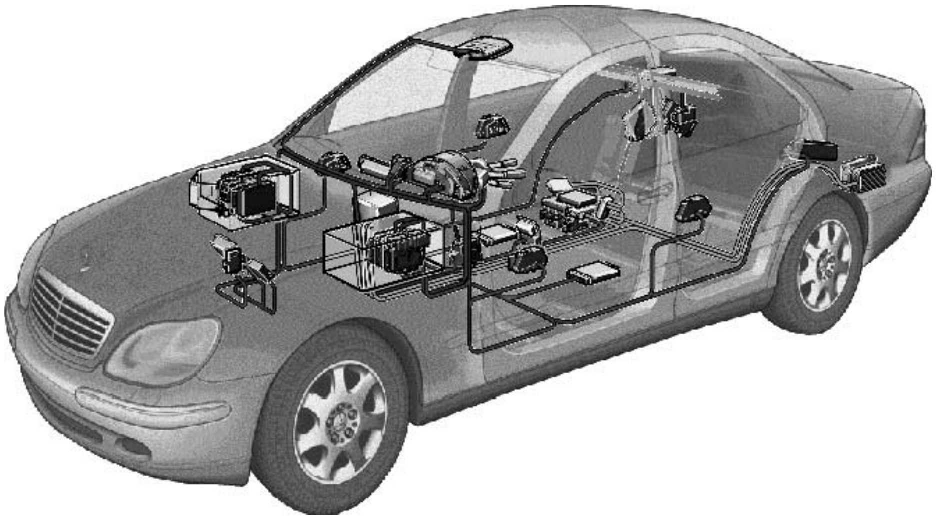


FIGURE 16.3 Sketch of ECU localisation in current S-class (Courtesy of Daimler-Chrysler AG)

The majority of the software is implemented by external suppliers, leaving us as a car manufacturer with the responsibility of specifying individual systems. ECUs are often purchased as ‘black boxes’ (thus incorporating both hardware and software). Of course, all rules have exceptions. Some parts of the software are developed in-house or by third-party software vendors.

This means that we as manufacturers have to orchestrate many different suppliers to realise one complete system. The specifications must ensure that the ECUs inter-operate properly. This means that the specifications have to define all ECU interfaces in detail, with requirements about interfaces, timing behaviour and error cases, described down to bits and bytes, protocols, and milliseconds. Otherwise, smooth integration of ECUs from different suppliers would not be possible (*see also* Weber and Weisbrod, 2002).

The period from initial requirements writing for a new vehicle model to the start of production ranges between three and six years in length. Given such a long period, changes are inevitable during development. There are many reasons for change:

- new features and functions released by competitors;
- technical feasibility and technological advance;
- changed legal regulations (e.g. fuel consumption and emission);
- expected savings in weight, space, or money.

The last of these, especially, makes the dependability and cost of hardware really important. Owing to the high number of units produced, savings in the region of a few cents per manufactured ECU may justify additional software development efforts in the range of several man-months. If, for example, there is a new sensor available that saves 10 cents per ECU compared to the old one, it may make perfect sense to switch to the new sensor even if a whole set of requirements has to be revised, causing additional down-stream development work.

Development of a new ECU is usually not ‘green field’: most often, there are comparable systems from earlier versions of the same model, or from sibling models. For example, the new S-class door control unit might be based on the E-class door control unit specified a year earlier. Consequently, the respective specifications are not created from scratch but by copying the relevant sections from existing documents. Having in mind that an individual specification document may consist of several hundred pages, consistent and complete copying and editing is a non-trivial task (Heumesser and Houdek, 2003; Alexander and Kiedaisch, 2002).

The specification of an individual ECU may be characterised as an extended negotiation process involving numerous stakeholders. The specification’s author has to triage all the raised requirements, to detect conflicting requirements, and to initiate the necessary negotiations. The author is also the primary contact person for all questions and issues regarding the new ECU. Thus, a specification author usually does not consider himself a requirements engineer. Instead, he considers himself more as a domain expert. He sees solution finding as one of his primary activities. Writing a technical specification is a necessary step in this process. Aligned with the self-image of domain experts, the involved people are often electrical or mechanical engineers; they are rarely software specialists.

STORIES IN AUTOMOTIVE SOFTWARE DEVELOPMENT

Keeping up good communication is one of the most vital challenges for present day development projects in the automotive industry. Different roles inside and outside the organisation have to interact smoothly to create a product. Marketing and sales gather information about customer profiles and needs—which have to be communicated to development units. Developers responsible for particular functions have to communicate and reconcile their requirements with developers responsible for adjacent functions, with the architecture group, with the user interface design group, with marketing and with others.

Research efforts (e.g. Schank, 1990; IBM, 2001) show that stories are one means of managing knowledge, that is, of storing and spreading ('socialising') it. This observation can also be seen as a justification of the intensive use of story cards in XP projects, for example (see also Chapter 13, *User Stories in Agile Software Development*). Since requirements engineering is a special kind of knowledge management, stories can also be applied in this area. In our practical work on improving requirements-engineering processes at DaimlerChrysler, we found that stories could indeed help: we tried out stories in several pilot projects.

By 'stories' we mean narrative, sketchy, incomplete, example-like scenarios as opposed to fully analysed, comprehensive, generalised use case models. Alexander and Zink (2002) describe this range in more detail, and discuss where in the systems engineering process different scenario methods are best applied (see also Chapter 3, *Scenarios in Requirements Discovery*, and Chapter 4, *Scenarios for Innovation*).

In the following sections, we illustrate the use of stories in the specification of automotive software. As the technique of writing 'stories' is a natural one, we do not describe the approach in detail. Instead, we focus on the context in which stories were used, rationales for our decision to use stories, and the effects we observed.

The adoption of stories in DaimlerChrysler is proceeding gradually. Some groups have embraced the idea and apply stories regularly. Others are just starting to try them out, and yet others have not come in touch with stories at all.

Stories in Requirements Discovery, Negotiation and Communication

So let's take a closer look at typical tasks when developing embedded software systems in automotive industry, and see how stories can be applied.

One such task is defining actual user needs gathered by marketing or sales. How do you select features to include in a new product, or a new version of a product? The features should correspond to requirements—needs or wants of people who will buy and use the product. These people are different from the people who will develop and test the product, so there is a need for discovering and communicating the requirements in a form that can be understood by people with different backgrounds and roles.

One way of sharing such user needs is to tell stories. People immediately understand stories. A story begins with a first step, and continues to an ending. This means that stories are good at showing end-to-end sequences of interactions between users and the system. In particular, working with stories when discovering and negotiating requirements helps engineers focus on user benefits rather than on details of technical solutions.

Stories enable people to share an understanding of user needs.
Stories help people to focus on user benefits, instead of on technical solutions.

We wanted to bring two internal parties of our organisation closer together with respect to developing requirements—let's call the first 'Marketing' to indicate that this party has a good understanding of end-users. The second we'll call here 'Telematics Development' to show that these are technology experts responsible for defining and developing product features. The goal here is obvious: to transmit knowledge about users to Development, and to reach a shared understanding of wanted features. The basic idea was to conduct a joint story workshop, and to set up a process for handling the stories further.

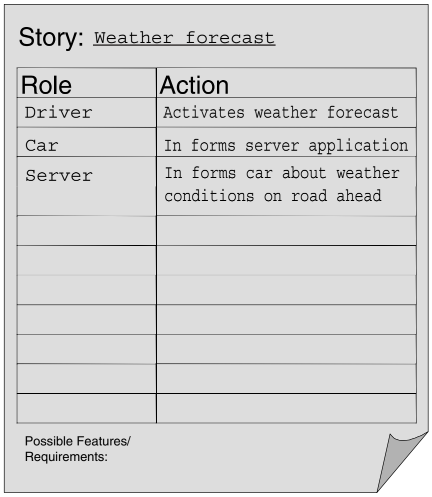
The design of such a workshop should be as simple as possible. For our purposes, the following proved to be practical:

1. Set up workshop goals and introduce participants to the workshop design.
2. Present an example story to explain the method.
3. Divide the whole topic into several subtopics.
4. For each subtopic:
   — divide into groups of 2—4,
   — each group writes stories for a defined time span;
   — after that the groups rejoin together, each group presents a story to the whole workshop (rotationally, one story each group);
   — finally, participants make a coarse clustering of the stories.

A similar workshop pattern is described by Ellen Gottesdiener in Chapter 5 (*Running a Use Case Workshop*). Of course, the group of storywriters should always have members of both parties, to ensure that they share their different expertise and attain a common understanding of product usage.

To focus on the interactions between the user and the system, we used a simple paper template, as shown in Figure 16.4, to collect the stories. The example application is an intelligent weather forecast system that informs drivers about bad weather conditions on the planned route ahead.

Besides the story's name, the story writers told the story in the form "⟨role⟩ does ⟨some action⟩",
which forced workshop participants to focus on interactions. To illustrate this point further, we also invited the participants to actually 'play' the stories through: for example, one participant could play the driver, another the telematics system, a third the mobile phone system. Obviously, this feels unorthodox at first, but again helps focus on interactions and user goals rather than on internal system details and technology. Thus, we confirm Suzanne Robertson's suggestion for playing through scenarios (Chapter 3, *Scenarios in Requirements Discovery*) and Gottesdiener's role-play approach (Chapter 5, *Running a Use Case Workshop*).


Story: Weather forecast

Role	Action
Driver	Activates weather forecast
Car	In forms server application
Server	In forms car about weather conditions on road ahead

Possible Features/
Requirements:

FIGURE 16.4 A simple template for writing initial stories in a workshop

Nevertheless, if in a first discussion about a story, for example, 'Making a bank account transfer with the telematics system', some early technology requirements and features pop up already, like 'Secure Sockets Layer-Support', which shouldn't be forgotten, the workshop participants have a little space to note these at the bottom of the template.

Stories in Requirements Analysis

Many details of our specifications today already exist, because there are some real-world situations in which these details play a vital role. Consider for example the wiper system requirement (see also Figure 16.5):

> *req-0001: The washing function for the front wiper shall be put off, if both the car's speed is less than 5 km/h and one of the front doors is open.*

At first glance, the rationale for this requirement might not be too obvious. But thinking more about it, one concludes that this requirement does in fact protect the passengers from getting wet, if one of them were to get out while the washing function continues. On the other hand, why is there a speed condition there? Well, consider the following situation:

> *One of the front doors is not totally closed. The driver doesn't notice this and drives as usual. A lorry driving next to the driver's car runs through a huge mud slop and causes many mud splashes on the car's windscreen. Unfortunately, the sun is low, shining from the front, so that the driver can see almost nothing through the windscreen. In this dangerous situation, he immediately needs the wiper's washing functions to clear the windscreen.*

By reading the story it should be clear why the engineer has thought about the speed condition (less than 5 km/h) in the requirement req-0001. But now suppose that this first

FIGURE 16.5 Wiping system of Mercedes C-class (Courtesy of DaimlerChrysler AG)

engineer takes other responsibilities, and somebody else takes over the specification of the wiper system for a new car model. The new engineer is young and dynamic, and always searches for improvements. When he reads the requirement req-0001, he might conclude—as we did—that this is to protect the passenger from getting wet. He thinks for a while, and concludes that in the document right now there isn't a requirement to stop washing if the sunroof is opened. So he includes such a requirement, but doesn't include the speed condition, as this isn't obvious to him. What he comes up with then is a wiper system, as in the story, which might not allow for washing the windscreen in a situation in which this is immediately needed (because of an open sunroof). He wouldn't have done so if he had been given the story mentioned above, as from this specific story everyone can easily generalise to similar situations.

Stories help protect requirements from deletion or well-meaning editing.
More generally, stories provide a pattern for new requirements (probably related to existing ones) to follow.

This tale is invented but quite close to industrial reality. Our experience suggests that by making these possible real-world situations explicit, they become much better thought through and serve as a basis for further requirements analysis (i.e. asking questions like: 'ah, but is 5 km/h a reasonable limit then? I could imagine the situation where [...]'). Obviously, stories cannot solve all the issues associated with requirements analysis, but they are an important piece in the puzzle.

This approach also brings us to the area of dealing with defects (e.g. a broken wiper) or system misuse (e.g. invoking a wiper cycle while the wiper is mechanically fixed). Defect detection and defect handling often represent more than 50% of the system's functionality. To our experience, stories are also beneficial in detecting 'what may go wrong'. Here, Ian Alexander's approach with misuse cases might provide some additional benefits (see Chapter 7, *Negative Scenarios & Misuse Cases*).

Identifying Feature Interactions and Conflicts

A car consists of several clusters of functionality: the body control (e.g. interior lights), the chassis control (e.g. ABS), power train (e.g. motor control) and telematics (e.g. navigation). Each of these coarse clusters has many functions, of which again each divides into several features and sub-features.

Different teams are responsible for different features, as different domain expertise is needed and work has to be divided. Historically, different features have usually been quite independent—the functionality can be specified in detail without too much knowledge of other features. However, in today's premium cars with highly integrated functions, complex kinds of interaction between features are nevertheless possible.

For example, suppose there is a feature in the telematics system that provides Traffic Warning Waiting: if a telephone call is in progress, a beep is heard and the Traffic Warning is delayed until the telephone call has ended. Now suppose that another caller tries to contact the driver, who has set up Call Waiting. The driver hangs up. Do the Traffic Warning and the Call Waiting collide? If so, what should happen?

Figure 16.6 shows the role of stories in this situation: the feature specifications can be thought of as columns, which are initially independent. A story can now involve several features in the steps of its narrative flow of events, such that the features involved are woven together by this story. The story then shows the interplay of the features, which might in the first iteration only be a question raised by one of the people responsible: 'What actually happens in this situation?' With a story, he has a means to point to a possible problem, and ask the other stakeholders what they think should happen. Together, they can rewrite the story to specify the wanted and agreed behaviour, and each of them can adjust the parts of the feature specifications to reflect this desired behaviour. One can view this as a pragmatic process to identify feature interactions, negotiate them, and document the reconciled solutions.

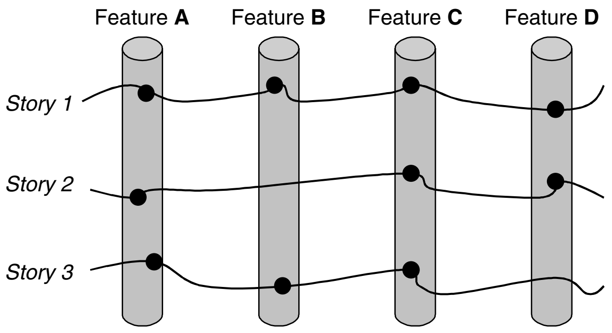


FIGURE 16.6 Features can be thought of as 'Specification Columns', and stories as means to weave features together, pointing to feature interactions or conflicts

Stories help people identify and resolve feature interactions.

Since features that are the responsibility of different teams may interact, it is essential for the development of a good quality system that a system-wide vision is shared among all the teams. Formerly, the number of possible interactions between subsystems in a car was limited. As the sub-systems become more 'intelligent' and have more inputs and outputs, the likelihood of adverse interactions rises rapidly. The cost of resolving these interactions may become large if these are discovered late in development, or when the system is actually in service. Therefore, it is important to discover interactions early.

Relating Stories to Features

We have shown above how stories can be used to develop and communicate requirements, or to identify and document feature interactions and conflicts. By doing so, that is, by writing stories, you get a bunch of stories you have to incorporate somehow into your existing development process and requirements documentation. The best way of doing so is to put the stories logically 'before' traditional feature specification (see Figure 16.7).

In such a way, the stories become firstly a justification of features: they are rationales for the features being there or for the feature specification to be exactly as they are (e.g. in case of feature interactions). Secondly, the stories are an index to accessing and understanding the feature specifications and in turn the system requirements. System requirements might be seen as structured models, and as Karen Holzblatt argues in Chapter 10 (*Role of Scenarios in Contextual Design*), there has to be a relation between sequential scenarios and models. If feature specifications are already strongly solution-oriented, stories may help engineers to maintain an overview of stakeholder needs.

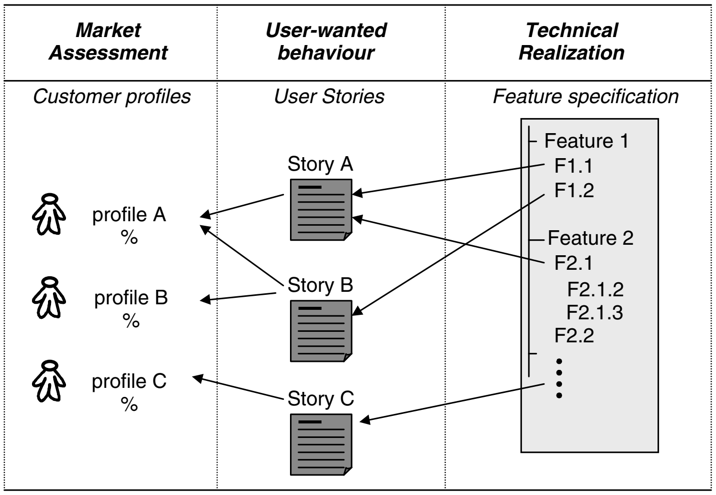


FIGURE 16.7 Relating stories to features

Stories help justify proposed features and requirements.
Stories help elicit requirements (that may have been assumed to be obvious).
Stories can be used to index features and requirements.
Stories provide engineers with an overview of stakeholder needs.

Stories describe wanted behaviour. But as Figure 16.7 depicts, we suggest that stories themselves need to be justified. Why do we think so? The reason is that products serve different market segments. In one such segment, the typical user (let's say that they are 80% of this product's users) might be business managers who need to have all kinds of communication channels with as many interfaces and options as technologically feasible—whereas in another product this kind of user might only make up 20% of all product users, but the user class of retired elderly ladies who need to have simple easy-to-use functionality might form a large group with 50% of all users. By relating such customer profiles to user stories (the relationship is many-to-many), the stories can be prioritised in terms of market needs.

Stories describe wanted behaviour.
Stories, being end-to-end descriptions at user level, are the right kind of structure to prioritise (all the requirements needed in a story must have at least that story's priority).

If, on the other hand, the development costs of features are assessed, product management can then weigh the benefits of 'giving the stories to the customer profiles' against the costs of developing the features needed to realise these stories. This should illustrate that keeping traceability between customer profiles, stories, and features eases feature prioritisation.

Requirements Recycling by Reusing Stories

In the common RE literature (Robertson and Robertson, 1999; Wiegers, 1999) the end user is thought of as the main source for requirements for a new product (see also Figure 16.7). Of course, this is in a way also true for the automotive area. However, we can distinguish two main types of embedded subsystems being developed for a new car model:

a. innovative functions not available before,
b. functions already being developed several times for existing car models.

In case (a), sub-system development is usually technology-driven, that is, domain experts try to figure out what is technologically feasible and at what development/production cost. To do so, they develop simulations and prototypes. In other words: the requirements there mainly come from the domain experts and not from users or business goals—although

of course these are often implicitly there (e.g. ‘the new innovation xyz will attract users by providing more safety; this will lead to higher sales”).
In case (b), many requirements are taken over from an existing requirements document. For example, if a specification for a windscreen wiper-subsystem has to be written for a new car model (in order to contract this out to a supplier) the new specification usually will resemble that from an existing car model—after all, the basic functional requirements for wiping haven’t changed for years. However, there might be differences in detail (e.g. a communication signal from another electronic control unit has changed, or for the new car model, the user should be able to choose between three wiping speeds, when up to now he was only able to choose from two wiping speeds). It’s obvious that type (b) development happens more frequently, so it’s a typical task for engineers to recycle existing requirements, which should be supported by the requirements-engineering methodology as much as possible.

Also in this situation, stories can be applied with benefit to foster reuse of existing requirements. This is because stories with their user-facing, narrative form are comparatively stable: for example, the story above describing that a hazard can arise if the wash functionality of the wiper system is not available for some reason, seems valid for as long as the windscreens and wiper systems we know today will exist. Of course, the underlying technology may change, communication signals may change, or sometimes even some functional details may change. But the stories are relatively unaffected by these changes as they shouldn’t go into such detail if written properly.

What’s more, if a domain expert needs to reuse existing detailed technical requirements, he first needs to have a good rough understanding of the basic situations that are important and from these he needs fast and easy access to the details. Now if the stories are connected to the technical specification by traceability links as, for example, shown in Figure 16.7, the story will serve as an index in the recycling process for existing requirements. This is because the domain expert can scan the stories (from which he gets a good initial understanding of the functionality, which might have been written by some other domain expert). He can then decide which stories are still valid, and for these he can follow the links to the technical details and see which of these details are also still valid, and copy them into the new specification. Also, if there are stories that aren’t valid any more or that have to be adjusted or enhanced, he knows immediately by following the links which technical details might then be obsolete as well.[1]

Stories are conveniently stable units of requirement reuse, whereas individual (atomic) requirements are vulnerable to technological change.
If requirements are carefully (accurately and completely) traced back to stories, those stories form a means of recycling requirements in subsequent products.

If the stories are not totally stable, how do stories themselves get recycled and how does one start with such a recycling process? Figure 16.8 shows the simplest approach

[1] This is however an optimistic view, as it assumes sufficient quality (completeness and correctness) of links between scenarios and technical specifications.

to starting and living such a process: for the first product (car model A), initial stories are written only for new functions. Besides that, stories might also be written ('re-engineered') for existing functionality, if some situations are unclear and need to be analysed further. In this way, the quantity of stories grows while developing (sub-systems of) a new product. If a parallel development effort for another product (car model B) starts while the development of car model A is still in progress, the current set of existing stories can be reused by copying these together with the detailed technical specifications and the domain experts can start the recycling process as described above (by using the stories as an index).

Copying is extremely flexible—the new project can work independently from the other development, but it automatically creates redundancy, which can lead to inconsistencies. For example, if a story is revised during the development of model A, it ought also to be changed in model B. Of course, with tools and traceability links, this problem is mitigated. On the other hand, one might also strive to make the stories product-independent and manage the set of stories together for all product instantiations. Whether and how this can be done efficiently in practice is one of our current research topics.

LESSONS LEARNT

'Engineers think in solutions, not in problems'. Generally, this is a good thing because solutions are the driving force for innovations. But focusing on solutions means that the problem can be forgotten as soon as a solution is found. But problem situations put solutions in context. Without context, it is hard to judge the appropriateness of a solution, or to decide whether to search for another.

Let us recall the wash-wiping example. Is it relevant that we have the speed limit 5 km/h, or could it equally well be 4 km/h? The story tells us that we just need a threshold to differentiate a running car from a stationary one. If the speed limit is relevant, there may be another (technical) story that tells us problems with having too low a threshold (maybe caused by the resolution of the selected sensor).

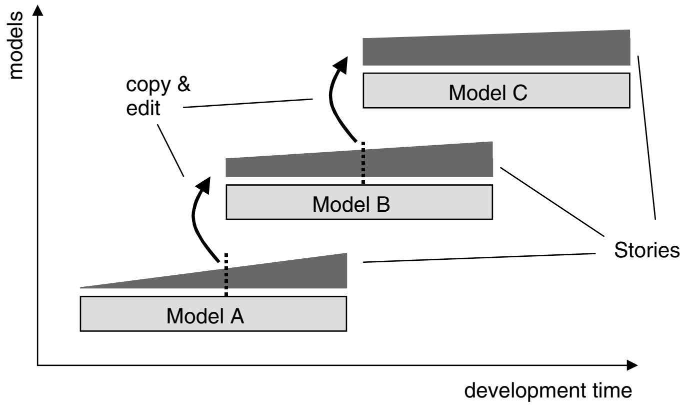


FIGURE 16.8 The simplest approach to recycling stories for different car models

Thinking in solutions does not only mean that the context is forgotten after a solution has been engineered. It also often means that the problem is not thought through thoroughly. Work may start from a rough idea such as a customer's suggestion, and a solution may be proposed—but the benefit to customers may not be evaluated. Using stories to document initial ideas automatically results in descriptions that combine user and real-world behaviour with system reactions. If stories are written down in a structured way (as sequences of situations and actions), they provide a good starting point for exploring alternatives; for example, 'what should happen, if wash-wiping is activated and the driver stops the car. Should washing stop?' Asking for alternatives and exceptions thus leads not only to more complete specifications but also immediately provides a set of test cases (see also Chapter 14, *Use Cases, Test Cases*).

So stories do help in many ways. But asked for our greatest lesson learnt we say frankly: 'be cautious!' Stories offer a method of describing dynamic behaviour informally. Their role is to improve communication, not to demonstrate completeness or consistency. Of course, they can be transferred into (or supplemented by) analytical, engineered use cases, which aim to be correct, consistent, complete, unambiguous, and so on. In other words, there is a continuum of methods available, ranging from the informal and concrete to the formal and abstract.

In our practical work, we experienced quite a few different project types with respect to stakeholders involved, complexity, innovation-grade, safety-relevance, system-level, and so on (compare e.g. Cockburn). What we discovered is that believing in just one of the methods and applying it blindly to different projects the same way is a great mistake. Only the opposite—analysing project type and needs beforehand and choosing from the variety—leads the way to more successful projects. The good news in this situation is that there are actually some project patterns that give hints on what methods to apply. We shortly present here two of them, which we believe are quite characteristic for the engineering of embedded systems:

The 'let's do it again'-project usually consists of a small team of engineers (a homogenous group of domain experts) responsible for some subsystem, for example, an ECU. Similar sub-systems, with some variations in functions and quality, have been developed by the team many times already. Team members understand each other well and they have an ongoing and deep discussion with their suppliers. Developing a new sub-system is mainly a matter of choosing the right features from the existing solutions, adding and changing some minor things to optimise the solution (e.g. altering some threshold) and putting this into a concise and consistent specification for development sub-contractors. As such 'let's do it again'-projects are strongly technology-oriented, and need a method of structuring the specification well to support intensive reuse. In this situation, a hierarchical feature-oriented approach serves well, and it's perfectly acceptable simply to supplement the feature specifications with stories or use cases in a few places to describe specific problems behind the specification, or to show and clarify some feature interactions.

A neat example is the development of instrument clusters, which are part of every vehicle. The basic behaviour of an instrument cluster is well known: there are precise specifications and competent suppliers available. Over time, the domain experts have built up their own 'domain model' with their own terminology and ways of expressing particular information. But, as in every environment that has evolved naturally over time, it can happen that some requirements were stated by people who had left the group.

For instance, at some point in time, we faced the situation that we were negotiating warning messages that a new instrument cluster should cope with. So we went through the already specified ones and identified which are relevant for the new car and which are not (e.g. if the new vehicle does not embody a convertible top, we can omit the related warning messages). In this situation, we found a message that no one could explain immediately. It took the engineers sitting around quite a while to re-engineer the underlying real-world scenario behind this message from the precise (but completely technical) specification at hand. Of course, it is a wise idea to write down this re-engineered scenario as a story, a use case or whatever fits the audience, and to maintain it with the technical specification.

Stories can serve as a 'corporate memory' for design decisions.

On the other side, we face 'great innovation' projects, that is, projects that try to engineer a completely new system. These projects can be characterised as follows: The starting point is usually a rough idea—a vision—for a new functionality or service. Technical details as well as user benefits, however, are completely unclear and it is the duty of the involved experts to figure them out. The involved experts are usually from different disciplines with different backgrounds. Sometimes, they may have worked on systems that might become part of the new system (which means that they have a certain idea of their work, which might not be easily integrated in the new context).

In such projects, we found stories that try to draw 'holistic' pictures of new systems useful. As we said, engineers think in solutions, not in problems: they are often tempted to dive into technical details before the overall picture is clear. We advocate a layered approach. First, there should be an agreement on the vision and scope of the new system. Then, we need to document the high-level requirements as completely as possible (without thinking about any details). After that we need to write stories for all areas raised by the high-level requirements. These stories should not be analytic, in the sense that every alternative and exception is spelt out in detail. Instead, the stories should point to every region of the area of interest.

Let us consider the example of a driver monitoring and emergency braking system. The basic idea is to monitor the driver's attention with a video camera. If the system detects that the driver is not paying attention to traffic, the system first tries to wake-up the driver; if this does not work, the system stops the vehicle in a controlled manner. Such a project probably needs to bring together experts from vision recognition, safety analysis, driver behaviour analysis, psychology, vehicle electronics, and so on. Such a team will work in depth on recognising inattentive drivers or initiating emergency braking. But the team may well not think about an 'after-braking' scenario. What will happen after emergency braking has been initiated? What will happen if braking was initiated on a steep hill? Is the activated brake able to halt the car there for a prolonged period of time, or might the brake gradually release? Informal but concrete stories can effectively help in the exploration of such situations.

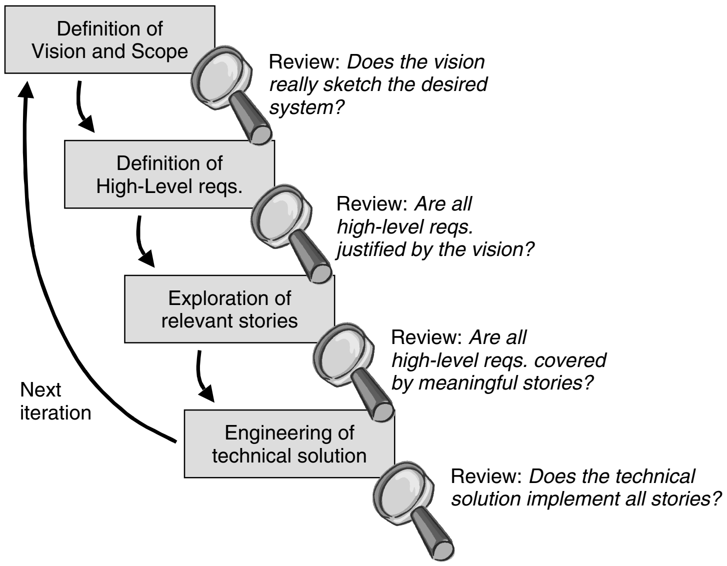

Definition of Vision and Scope
Review: *Does the vision really sketch the desired system?*
Definition of High-Level reqs.
Review: *Are all high-level reqs. justified by the vision?*
Exploration of relevant stories
Review: *Are all high-level reqs. covered by meaningful stories?*
Engineering of technical solution
Review: *Does the technical solution implement all stories?*
Next iteration

FIGURE 16.9 Iterative refinement

> Stories help explore side effects, consequences, and easily forgotten situations that need to be designed for.

Another characteristic element of such 'great innovation' projects is that they have to be iterative. It is unlikely to do the job right the first time. We might have to figure out that something that seemed simple at scenario level turned out to be more complicated (or impossible) at the technical level. It might also be that a solution at hand (e.g. a working prototype) causes new or changed visions. We found it essential to maintain traceability between the various levels of abstraction. This can be achieved with a rigid process of definition and analysis, as shown in Figure 16.9.

All in all, stories provide a useful vehicle for supporting the engineering of embedded automotive systems. But their application has to be justified by actual needs in development projects.

We should mention one last problem: Stories do not look like proper engineering artefacts. Stories are informal and imprecise—making them look completely unlike familiar artefacts such as state charts or pin descriptions. This can make engineers sceptical of stories when they first come into contact with them, so story workshops need to be properly facilitated.

KEYWORDS

Narrative	Embedded Systems	Feature Interaction	Requirements Documentation
Stories	Features	Requirements Negotiation	Automotive Industry

REFERENCES

Alexander, I. and Kiedaisch, F., Towards recyclable system requirements, *Proceedings of the 9th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS 2002)*, Lund, Sweden, April 2002, pp. 9–16.

Alexander, I. and Zink, T., An introduction to systems engineering with use cases, *IEE Computing & Control Engineering Journal*, **13**(6), 289–297, 2002.

DaimlerChrysler, Soft and Safe, DaimlerChrysler Hightech Report, 1/2002. http://www.daimlerchrysler.com/research/htr2002/pdf_e/elektronik_6_e.pdf

Davis, A., Requirements Engineering, in J.J. Marciniak (Ed.), *Encyclopaedia on Software Engineering*, Vol. 2, 2nd ed., John Wiley & Sons, 2002, pp. 1145–1155.

Grimm, K., Software technology in an automotive company—major challenges, *Proceedings of the 25th International Conference on Software Engineering*, Portland, May 2003, pp. 498–503.

Heumesser, N. and Houdek, F., Towards systematic recycling of system requirements, *Proceedings of the 25th International Conference on Software Engineering*, Portland, May 2003, pp. 512–519.

IBM Research, Fostering the Collaborative Creation of Knowledge—A White Paper, John C. Thomas, IBM Research, Hawthorne, 2001, http://www.research.ibm.com/knowsoc/

McKinsey, Automotive Software: A Battle for Value, Preeti Bendele, Andreas Cornet, Guido Haak, Bernd Heid, Ulrich Näher, Klaus Richter, Auto Software, 2002, Vol. 11.

Robertson, J. and Robertson, S., *Mastering the Requirements Process*, Addison-Wesley, 1999.

Schank, R., *Tell me a Story: Narrative and Intelligence*, Northwestern University, Evanston, IL, 1990.

Weber, M. and Weisbrod, J., Requirements engineering in automotive development: experiences and challenges, *IEEE Software*, **20**(1), 16–24, 2003.

Wiegers, K., *Software Requirements*, Microsoft Press, 1999.