Software Design Reviews

December 31st, 2009

Nora Ludewig
(nludewig@gmx.de)

Introduction

This paper describes patterns for software design reviews.

The experiences leading to these patterns were made in the context of software development for embedded systems. However, they contain practices that can be useful in every kind of software design review. I became aware of these patterns while I was learning to apply the review method DRBFM to software development. DRBFM stands for Design Review Based on Failure Mode and is a review method invented by Toyota and now slowly spreading into other organizations. Initially, it was developed for proving the robust design of mechanical products. But it turns out that DRBFM is a powerful way to look at software designs, too.

DRBFM focuses on the separation between functionality and solution. The method helps you to check whether a product still fullfills the requirements in a robust way when you implement changes in one or several parts of the product. Interactions between different parts are looked at in particular.

Software for embedded systems typically is connected very closely to the "real world". There are sensor signals that have to be transformed into numerical values, maybe the system has a processor working with integer arithmetics, and the system always has time constraints because there are limited reaction times specified. Due to this close connection of software and hardware there are many constraints on the software such as accuracy of the conversion of sensor signals or runtime and memory consumption. In the design review the software must not be analysed as an isolated product but as one part of a system.

In the field of embedded software development, reviews are often held after every development phase. There are reviews of the specification, of the functional design, of the code, of the software and system tests. All the different review methods are based on the assumption that it is helpful if someone different from the developer examines a product and brings in their expertise. Therefore the main parameter defining the success of the review is

the knowledge, the communication and the behaviour of the review participants [1]. This paper describes how you as a developer can contribute to the success when you are preparing, organizing and moderating a software design review.

In some of the patterns, I talk about "changes" or "modifications". This results from the approach of DRBFM always comparing a product as it was before with a new ("changed" or "modified") solution. So I assume there is always some basic version of a product, artifact or piece of code to start with. We are not re-inventing the software from the beginning, but there is some comparable product or some frame we can start our work from.

The thirteen patterns are divided into four groups:

Deciding what to review	Risk over Size
Preparing the Review	Continuously Prepare the Review Review the Final Design Don't Think – Describe your Solution! Make Implicit Requirements Explicit Look at the System Free Requirements from Solution
Running the Review	You are the Entertainer Every Question is Okay Do not Justify, just Listen Do it your Way – Don't stupidly fill in Templates It is not about Form
After the Review	Make the Outcome Concrete

With these patterns I would like to address developers, engineers and students working in embedded software development. There is a strong focus on the developer who is preparing and running the review. Nevertheless they can be helpful to review participants, too. Also they could be interesting for managers who want to improve the quality and the output of design reviews performed in their organisation.

Deciding what to review

Risk over Size

Context: You are designing a new software part or a change in an existing part and a design review is required by the process.

Forces: In software development, not only the final product counts but also the development time and effort. In most cases, developers have to make a trade off between quality and adherence to deadlines. The availability of time and expertise or the mere size of a software module determine the content and the intensity of reviews. So a rather common algorithm might be looked at very carefully while a very innovative software part might be more or less neglected.
The developer of new software usually has some influence on the intensity and the extent of the review that is done on the design. But if he sets the focus improperly, maybe big risks will be overlooked.

Problem: When you have to focus your review effort due to time or capacity restrictions, which parts of the development should you consider in the review?

Solution: Find out where is the biggest risk and put most of your effort on this part.

Implementation: A design resulting in a high number of lines of code is not automatically riskier than a small change in an existing software part. You should watch out for changes with a high degree of innovation or changes in complex interfaces. [2]
If you have used a commonly known solution or you have developed a module similar to one you have developed before, the risk is not as high as there is for completely new algorithms or changes in interfaces with cross connections to many other parts of the software.
Of course it is difficult to find an impartial technique leading to decisions comparable to decisions in other reviews. It is hard to measure risk which does not only depend on the properties of the software but also on the properties of the developer and his or her environment. By discussions with colleagues about your development and how it impacts the system, you will develop a sense of how intense the review should be.

Examples: 1. You are using a filter algorithm looking very complicated at first sight. The code for it is rather big. But in fact you have a lot of experience with this algorithm because you have used it many times before. Moreover the algorithm does not have many boundary conditions or dependencies. Therefore the review of this part of the design does not have to be elaborate.

2. You are changing an interface used as an input for a timer from a 32-bit-variable to a 16-bit variable. Maybe you have tested the new function in the target environment and you have shown that the behaviour is the same as before. Unfortunately you did not consider the fact that now there is a variable overflow much faster and that then the timer will fail. This example shows that especially when there are changes in interfaces it is good to review with a rather high effort.

Preparing the Review

Continuously Prepare the Review

Context: You are planning all the different tasks of the design phase, including the design review. You want to allocate time for the review preparation.

Forces: The review often is perceived as a point in time. That is the point when the review meeting takes place.
During the design phase, you have to keep in mind all the requirements and constraints to your product as well as the process you have to follow. Preparing for the review is another task you have to plan for during the design phase. The review preparation will consume a certain amount of time and energy in addition to the core design tasks. For this reason, review preparation often is neglected.
If you take care of the review only very shortly before the review meeting, you will prepare and organize the review in a hurry, forgetting about important points, inviting the wrong people, neglecting presentation rules.
Consequences of these forces are unprepared reviews, unsatisfying results and unmotivated participants.

Problem: When should you start to prepare the design review and how should you allocate the preparation efforts?

Solution: Start preparing the review from the beginning of the design phase and continue all through the design process.

Implementation: Take notes about concerns or questions you have. Talk to the colleagues you would like to invite to the review. Prepare the information you want to give to the review participants before or during the review.
Set the date for the review meeting as soon as you know when you will have finished the design. REVIEW THE FINAL DESIGN and plan some time after the review for doing some rework if necessary. Early planning will increase the chance that you get the experts you want to have in the meeting and they will have some time to prepare for the review.

Example: You design a software module for the control of a machine. This module has the engine speed as an input variable. In fact there are two variables representing the engine speed available in the software. A raw signal NEng_unfilt and a filtered signal NEng_filt.
At the beginning of the design you are not sure yet which signal is best for your application. You are taking the raw signal for the first draft version. As you have started to prepare for the review from the very beginning of the design phase, you always have in mind that you need to clarify this point until the design is finished. So there is no risk that the first choice for the draft version will stay in the solution just by chance.

Moreover, when you present the input values at the review meeting, you can say which variable for the engine speed you have chosen and, very important, why.

Review the Final Design

Context: You want to set the time for the review meeting.

Forces: A review meeting held at the wrong time can lead either to discussions about what is really the concrete solution and by this to a kind of developers workshop. Or it can lead to a hurried meeting where the developer tries to dismiss as many points as possible in order to avoid rework.
If you hold the review meeting too early, there is a danger that not all the necessary information will be available or some details even are not defined yet. You will have to review alternatives or theoretical concepts. This can result in endless discussions.
On the other hand, if the review meeting is held too late, you will not have the time to implement the measures you have decided on in the meeting. Maybe you will even try to avoid some findings in the meeting because you know that will not have the time to do the rework identified to reduce the risks.

Problem: When is the best time to hold the review meeting?

Solution: Hold the review meeting only after you have collected all the facts the review participants need to examine the design but keep enough time after the review to change the design where necessary.

Implementation: As you CONTINUOSLY PREPARE THE REVIEW, you can look for the best time for the review meeting from the beginning of the development. You can already talk with the colleagues you would like to participate in the meeting so that you find out when persons that are very important for the review success are on vacation et cetera.
As the delivery date normally is fixed, you can estimate how much rework will be necessary after the review in the worst case and then you can calculate the date when the meeting has to be held the latest [3]. The review participants also need some time for the preparation of the review. Taking this into consideration you know when your design has to be finished. So by calculating backwards from the delivery date you make sure that you have enough time for the review and the rework and that you also go into the review with a real design and not only a design idea.

Example: Here you can look at the same example as in CONTINUOUSLY PREPARE THE REVIEW. Imagine you are still not sure yet whether you should take the filtered or the unfiltered signal as an input value. Therefore you do not put NEng_filt or NEng_unfilt into your design, you just write down NEng. If you are doing the review now, there is a high risk that some of the participants will have NEng_filt in mind and some will think about NEng_unfilt. They will be talking about different things without noticing the problem. If they notice this

open point, the risk is high that the review will not be a design review on the complete design but a discussion about which of the two variables would be the better one for your application. Of course this might be a helpful discussion but it is not the goal of the review.

Don't Think – Describe your Solution!

Context: You are preparing the information **about your design** you want to give to the review participants before and during the review meeting.

Forces: When developing software, you do not just write code. You also define new interfaces, your software consumes memory and clock cycles in the processor, you might call library functions, maybe you use new development tools. This is often overlooked. As a developer you tend to focus on the changes you consider to be important. Changes that do not affect the functionality are easily ignored. However, things that seem unimportant to you might affect the overall system or parts of the system interacting with your module.
But you want to avoid to drown important changes in unimportant "stuff" and you think that a comprehensive list of changes might overwhelm the review participants. So your idea is to make a selection of the most important aspects in advance.

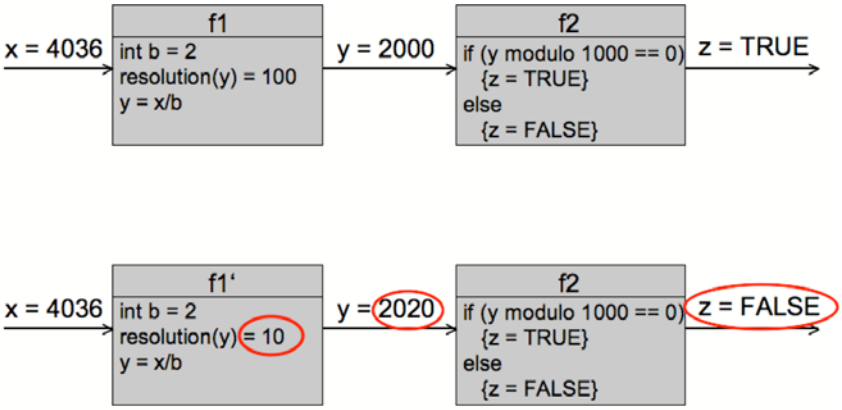
Problem: When collecting the information the reviewers need for analyzing your development, how do you decide on the relevance of the information?

Solution: When preparing for the review, describe every part of your solution, even if it seems unimportant to you. Avoid any judgement on the importance or the risks of individual aspects. Try to be as neutral as possible.

Implementation: A checklist can be really helpful to identify all the changes that were made.
If you have filled out a checklist before, you can show this checklist to the review participants. If there is a very high number of changes, you can, later in the review meeting, make suggestions to put some details aside or summarize some minor changes in one common heading. But you have to justify why it is safe to do so to the review participants.
If safety is a concern, you should rather err on the side of caution and mention too many changes rather than too few.

Example: The precision of the output value of a function is increased while the functionality itself remains unchanged. To the developer, this change seems to be unimportant, as his function is now providing a more precise value than before. Unfortunately, there is another function which uses this value as an input and performs a modulo operation on it. As now the resolution is higher, the results of the modulo operation will be completely different. The function fails.

If in the review this change had been mentioned, the potentially fatal consequences could have been found.



Make Implicit Requirements Explicit

Context: You are preparing the information **about the requirements** to the design you want to give to the review participants before and during the review meeting.

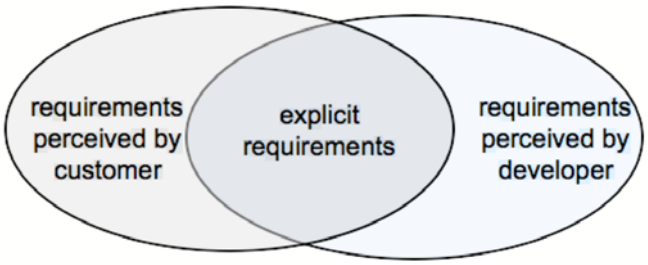
Forces: When judging whether a product meets the requirements everybody makes his own assumptions about implicit requirements
You have designed your new software module based on the specification of the customer. The customer wrote the specification from his point of view. Often, he has no information about the processes or constraints in your company. He might not know all the technical details of the system your software will be integrated into. But these details can have significant influence on your choices of specific solutions during the design phase.
On the other hand, the customer has his own processes and constraints. His company's philosophy might be completely different from the philosophy of your company, a fact that you, both the costumer and the developer, often are not fully aware of.
For these reasons you can almost be sure that the requirements are not complete from your point of view. You will design the software based on the requirements you got from the customer, completed with assumptions on what the customer wants to have but did not say explicitely.

Problem: How can you make sure that you as the developer, the review participants, and the customer have the same understanding of the requirements?

Solution: Specify clearly what you think are the requirements to the product even if these requirements were not mentioned explicitely by the customer.

Implementation: Put yourself in the place of the customer and think of what he or she wants the product to do or not to do. What are the different pos-

sibilities of using the product and how can they be combined? List all the aspects and circumstances of use. Think about what might be obvious to the customer but not to you. Then, think about what might be obvious to you but not to the customer.
You can take these assumptions as a basis for discussion. Chances are high that in the review meeting there is an expert who can confirm or correct these assumptions.
When you are wondering which requirements are obvious to the customer but not to you, of course it is a good idea to simply ask the customer. But even then, there will be implicit requirements remaining implicit if you do not find them and make them explicit.



When looking for implicit requirements it might be helpful to use checklists. In DRBFM for example, you are invited to find, for each module, the "basic function" which describes the actual purpose of the module, "additional functions" which focus on the on the look and feel of a product, "harm prevention" which avoids that the product can cause harm to people, or the environment and "self protection" which avoids that the module itself is damaged or destroyed.
Of course you should not wait until the design review to talk to the customer about the requirements. If there is information missing at the beginning of the design phase you should get answers to the open points as quickly as possible. Still, the design review is a good opportunity to present and discuss the requirements.

Example: The customer ordered a new application for a mobile phone: Photos taken with the camera integrated into the phone can be sorted into different albums. He has given you a specification where all the different possibilities of sorting, opening and modifying the albums are described. As it seems obvious to him, he has not specified the fact that the basic function of the phone, namely the possibility to place and receive calls must not be disabled by the new application. If you do not integrate this requirement into your review, you might only find out that your design is good for sorting photos and so fulfilles the requirements you limited the module to.

Look at the System

Context: You are preparing the information **about the environment** of your module you want to give to the review participants before and during the review meeting.

Forces: When you are developing only a small part of a complex technical product you tend to find a solution fitting this small part but maybe carrying risks for the whole product. If you do not consider this in the review you will not find these risks.
You know best about all the obstacles you had to overcome and about the reasons why you have chosen this specific solution. Digging deeply into the subject for a long time, you have probably lost an overall view on the product. As a consequence, there is a risk that you have created a locally optimized solution which works well if you regard it for itself but which does not really fit into the greater context. [4]
In software development, especially in large projects, there is a risk that a developer who is working on a specific part of the software for a long time loses sight of the real function of the product.
Moreover, in embedded systems, software is not an end in itself. Often there are many interfaces of one software function to other functions and to sensor signals or device drivers. It is difficult to keep in mind all interactions all the time during development.

Problem: Where should you make the cut when you have to decide how much of your software module's environment you should keep in mind for the review?

Solution: Ask yourself what is the function of your development for the whole product and document this function during review preparation.

Implementation: When preparing for the review it is important to step back and look at the whole system and at all the interfaces that your software module has with the overall system. These are inputs and outputs as well as environmental conditions.
Find answers to the following questions:

1. Why was there a need for this development?
2. Why did the requirement come up right now?
3. Have there been other attempts to fulfill the requirements? Why were they successful or why did they fail?

Example: The software is getting a sensor signal as an input and calculates some output value based on it. In order to find out if the output value meets the accuracy requirements you have to get some information about the accuracy of the input value. You read the technical data sheet of the sensor. But this is not the only information you should take into consideration. You should also ask yourself: Is it possible, in certain circumstances, that the sensor signal does not have the specified accuracy, although the sensor works properly? When might this be possible? For example, due to the signal processing from the raw

signal to the signal received by your module. Maybe there is some filtering or some hysteresis applied you did not take into your consideration. Also: what do I do (in my module) if the accuracy is not as good as I expect it? Can my module deal with that?

Free Requirements from Solution

Context: During review preparation phase, you are describing the requirements to your design.

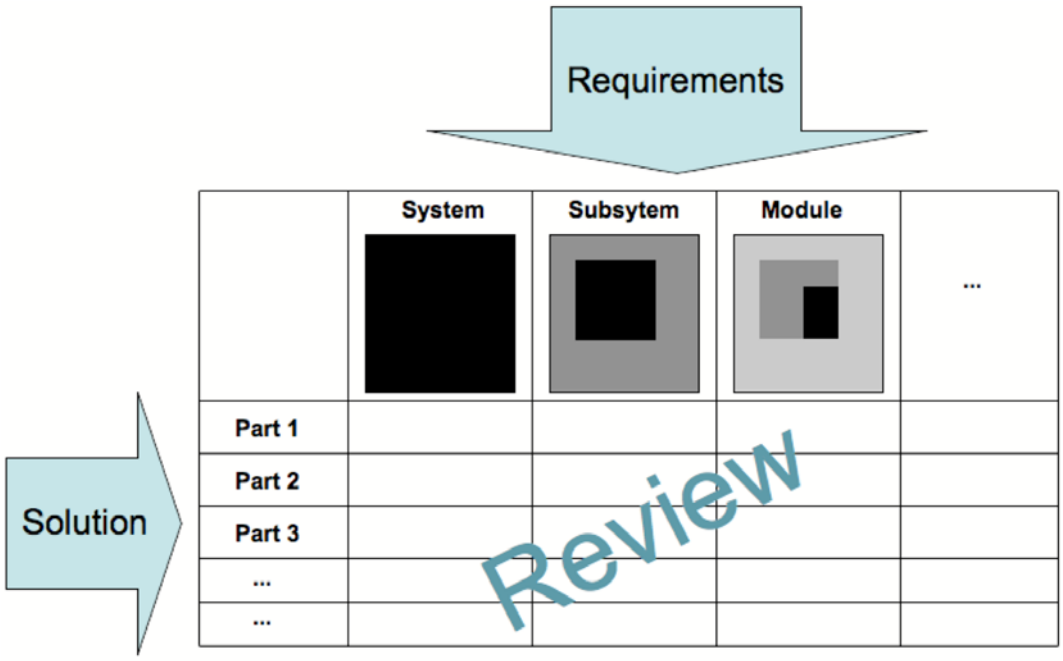
Forces: When thinking about the requirements, a developer never is unbiased. He or she always has the solution in mind. Thus there is a high risk that in the review there will be no clear separation between solution and requirements description.
The problem with this is that if you do not clearly state what your product should do you cannot tell whether the design is applicable or not. Also, if you and the other review participants think into a certain direction from the beginning, you limit the chances that you will find the best possible solution. The review should lead you back to the question what you really need this part of the system for.
Also, boundary conditions are essential because they define the scope you have to look at. Many designs work under certain conditions, but in the review, it has to be checked if these are the conditions that will prevail in the real system.

Problem: When you have to describe the requirements the design is to be reviewed against, how can you make sure that you are not biased by the design work you have already done?

Solution: **Ask yourself and your colleagues what your development should really do. Why was it requested by the customer, what does he expect it to do?**

Implementation: When describing the requirements, make your product a black box and describe its behaviours seen from outside. Divide your description into two parts:
First, describe the function. What is really the job of the module? How does the customer want your module to behave? What does the system your module will be integrated into "expect" from your module?
Second, describe the boundary conditions. This can be environmental conditions the product still has to work correctly in. It can also be requirements on precision, calculation speed or control quality.
If you have a rather complex design with different parts, it makes sense to break down the requirements into smaller pieces. In other words, you have to open the black box and make several smaller black boxes out of it. This sounds contradictory to the the pattern itself, but it is not as long as you keep the boxes black at each level of granularity you have chosen and you make sure that there is one black box at the same level of your whole design. Having this

highest level, you get the view on the complete design and the interfaces to the system.



Example: You are describing the functions of a software part modelling the temperature at a specific location of the system (virtual sensor). This software consists of several subparts. So the overall function would be "calculating temperature at position x". Describing the function of one of the subparts you might say "signal range check of sensor input S1". That means you are looking into your top-level black box "temperature calculation" and your are delimiting the solutions to those solutions using S1 as an input. The key point is that for this granularity level you must not say "signal range check with algorithm a" but only "signal range check with boundary conditions c1, c2 and c3".

Running the Review

You are the Entertainer

Context: You are moderating the review meeting.

Forces: Maybe some of the review participants see the review rather as an inevitable evil and so they come into the meeting without any motivation and without any expectation. They probably have attended a lot of boring review meetings before.
There is a big risk that, when some people with this attitude meet, this will turn into a self-fullfilling prophecy. They expect to have a boring meeting and so they make the meeting boring. This blocks all the creativity and concentration and finally leads to nonsatisfying results.
Even if your colleagues come to the review meeting with positive expectations, they can become unmotivated very quickly if you are presenting and moderating the review in a boring or confusing manner.
If you are not able to arouse the review participants' interest for your work, there will be not only a lot of time wasted, but there is also a higher possibility that not all the risks and errors will be found.

Problem: How can you motivate the review participants to contribute to a good review outcome?

Solution: Make the review fun! During the review meeting, you are the entertainer! You are responsible to create an atmosphere where all the review participants are comfortable and really want to contribute to the analysis of the product.

Implementation: At the beginning, you should make sure that all the participants know each other and know you. This helps everybody to get some orientation in the group.
Second you should give a short overview of the review so that everybody knows how long the meeting will take, what will be the steps to take and, very important, what should be the outcome of the meeting.
When you are presenting your work, tell a story to the audience. Give information about the motivation, about your ideas for a solution, about which obstacles you had to overcome, which actions you took in order to make the outcome robust and what is the final result. It does not mean that you should manipulate the information in order to create a breathtaking story or to show what a great developer you are. The story does not have to be thrilling and it does not need to have a happy ending. But is has to be clear, logic and complete. By this you make sure that the review participants can follow. Also it creates an open atmosphere and motivates people to account for the story in order to make it a succesful one.

Use the fact that IT IS NOT ABOUT FORM in order to arrange the meeting so that the participants remember it as some very interesting and motivating hours.

Example: You have designed a software module for an automobile engine control. For the development you had to do measurements at very low temperatures in a climate chamber in the middle of summer. So you came out of the climate chamber with a complete arctic dress in July.
In the review meeting, show a photo of you in your winter dress to the colleagues. Instantly they will pay attention and they will be inspired to be creative.

Every Question is Okay

Context: During the review meeting, you get a lot of questions from the review participants.

Forces: Maybe one of the participants asks a question you or a colleague consider to be trivial or irrelevant. As always, the time for the review is limited, and you want to get to the crucial points quickly. So you are maybe not answering all the questions elaborately.
But there is a high chance that there are more review participants who do not know the answer to the question and who just did not dare to ask.
If not every review participant gets the chance to understand the facts, they will probably have problems to follow the remainder of the review. You will not be able to benefit from their expertise. So you are wasting their time and the outcome of the review will probably not be as good as it could be.
Sometimes, we also tend to declare questions irrelevant, because we unconsciously know that we do not have a clear picture of that point in our mind.

Problem: How should you deal with questions you perceive to be superfluous?

Solution: Answer all the questions and do not (even not impicitely) declare them as stupid or the answer as obvious.

Implementation: You should not make excuses when questions are coming up. If you do not know the answer, be honest and suggest that you will find out after the review meeting. You should mention this task in the action plan in order to MAKE THE OUTCOME CONCRETE.
If you have the impression that the answer to the question is obvious to most of the other review participants, invite them to give the answer.

Examples: It is often the basic questions that cannot be completely answered: Are the assumptions made for defining the boundary conditions correct? Are we really sure to have chosen the best algorithm?

Do not Justify, just Listen

Context: You are in the review meeting and you are getting feedback about your work from the review participants.

Forces: When you are doing the review, experts from different fields will look on your work from many different points of view. Most of them will probably know that they were invited to the review because they are considered experts in their fields. So they will be proud to give their opinion and to ask you why you did it this way and not that way.
You will have a different perspective on your work. You might think: „How easy is it to question my solution. I have spend long nights at the desk, struggling for finding a solution at all". In other words, you might feel offended. So you prevent an open discussion and an open mind for the best solution.

Problem: How can you make sure that you get the maximum benefit out of the review participants' feedback?

Solution: Listen to the feedback carefully and make sure you are getting the point. Do not justify your work or presentation. Appreciate the questions and the advice as they will improve your product and your future work.

Implementation: If it was so difficult to find a solution and many boundary conditions have left you no choice, take the questions as a feedback. You should have illustrated the problems and challenges more clearly in your review presentation. Then the experts could have found out if their alternative solution really would have been possible.
Explain as long as there are issues that need clarification for the reviewers, but avoid discussing who is right or why you had no choice but had to do it that way. When participants are starting to repeat remarks and you have the impression that they just want to demonstrate their expertise stop the discussion and continue with the next point.
Give the review participants the feeling that you have got their point. If you have not, ask them to explain their question or remark again.
Keep in mind that, after the review, it is you who is deciding what to do with the opinions of the experts. So you might say, „This time, I won't change my solution as it is still robust. For the next time, I will keep in mind what expert E said."
So, a design review is not just a method to examine products for their robustness. It is also a way of exchanging ideas, spreading knowledge, building networks and improving communication skills.

Example: One of the review participants says, "I don't think that you have identified all the input parameters to your software model. In my opinion, input IN243 has to be added."
Your answer could be "Thank you for this idea! Could you please explain why you think that this is an important input? Do you know if in model m95 which is similar to mine this input is also used?"

Do it your Way – Don't stupidly fill in Templates

Context: In your company, you have to follow a certain review process. Maybe you have to take into consideration templates, checklists, or fill databases with information about your design.

Forces: As you want to fulfill the requirements of the processes, you might spend a lot of energy to force your review into the process.
A defined review process is necessary in larger companies, but it always carries the danger that developers become annoyed about all the administrative work they have to do. So the review participants will fill in all the templates and forms, but not before they have switched off their brain. The problem can also occur when template categories or questions do not match your context.
When you are trying to stick to the rules slavishly you might not be able any more to have a review where the real content and not the form is discussed. So you won't find the risks.

Problem: How can you avoid to be paralyzed by your organization's review rules?

Solution: **Adapt the processes and the templates to your solution and not vice versa. As long as you communicate this clearly and you can give clear the reasons, this is okay.**

Implementation: Maybe you can change the checklist or template categories, insert new rows, or columns. Adapt the review and its templates to your needs and not vice versa.
If you leave out process steps or some chapters in a template you have to give reasons for this decision. If you add something this is usually not seen as leaving the process but rather as an add-on, so you will not have any problems with this.
This pattern is not an invitation to cheat. Of course you have to stick to the rules. But if you bother to think about the templates and the meaning of the process steps you will find your way to use them as helpful support tools. Moreover you can give feedback to the process developers and help creating a leaner and more useful process in the long run.

Example: In your company you have to use a template where you have to fill out a chapter "changes in logic". In fact you have changed something in a model representing the physical behaviour of the system. You do not really find a category where this change fits into and you wonder if this should be put into the chapter "changes in logic". In order not to confuse anybody you add a chapter "changes in physical models" and describe the change there. In the chapter "changes in logic" you put a dash.

It is not about Form

Context Preparing for the review, you have collected all the different pieces of information about the requirements and boundary conditions. Together with your solution, you want to present them to the other review participants. Maybe you have also made some simulations or you even already have a running prototype of the software you are developing.

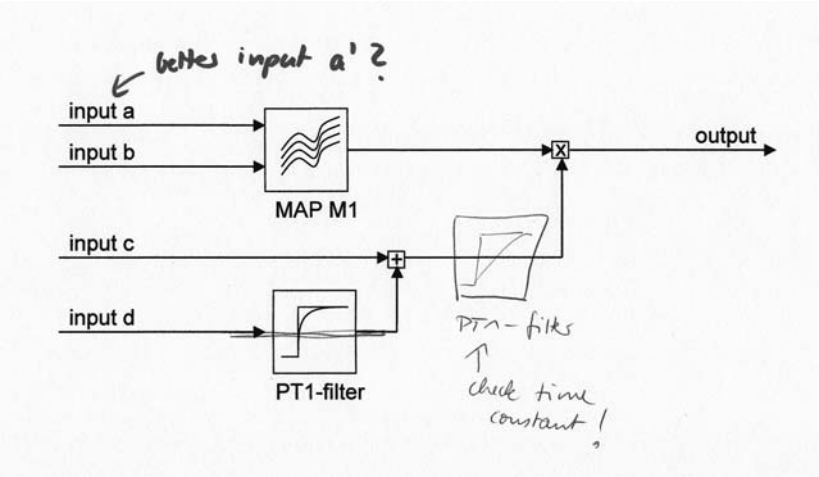
Forces: As YOU ARE THE ENTERTAINER, you want to have something nice to show to the review participants. You want to make sure that every review participant can contribute to the outcome of the review. So you maybe try to get as much information and demonstrations as possible to the review participants.
Structures and relations in large software projects are usually complex and not easy to understand. Even vizualisations which were very simple in the beginning can become confusing very quickly.
When you present your work in a perfect form with a lot of „special effects", it is more difficult for the review participants to really understand what you did and to get in a discussion. They have to „believe" because they cannot retrace your steps of work.

Problem: How should you present your work in order to facilitate a creative and open discussion?

Solution: Limit technical means and „show effects" to the really necessary. Do not waste your time with creating perfectly designed presentations, put the time into the SW design instead.

Implementation: It is good to give the audience a motivating start and to visualise as much as possible throughout the review meeting. But although YOU ARE THE ENTERTAINER, you should avoid making the design review a design show. Keep it as simple as possible.
Make drawings on the white board as you explain things instead of letting a power point slide suddenly pop up completely filled.
In many cases you can print out the facts and the figures you want to talk about, hang the printouts to the wall and then discuss with the other review participants while standing in front of them. This might seem to be old-fashioned and requires a little more preparation than taking the laptop to the meeting room, but it gives the review participants the possibility to really interact with each other. If they want to know a number you have presented ten minutes ago, they do not have to ask you to go back in the slides. They can look it up themselves in the printouts. Another advantage is that everybody can take out a pencil and show his ideas to everyone on the printouts. This can be very useful if he or she suggests a small modification in one of the figures.

Example:


better input a' ?
input a
input b
MAP M1
output
input c
input d
PT1-filter
PT1-filter
check time constant !

After the Review

Make the Outcome Concrete

Context: The review meeting is over. In the meeting you have found some need for rework, for example you have to get more information on a specific topic, you have to do some tests or measurement. You and your colleagues will have to accomplish these tasks.

Forces: Once the review team gets to the point where the need for validation or enhancement is identified, there is a great temptation to say „Well, we have found the critical point now, and we will think about what to do exactly about it later."
Maybe, you are even defining the necessary actions in detail in the review meeting, but you are only writing down some notes or keywords or you do not decide on a responsible and a due date.
Even if everything is documented still somebody could miss the information because he or she is not among the review participants.

Problem: How can you make sure that everybody can continue their work with the review outcome?

Solution: Give a detailed and comprehensible description of all the actions defined in the review. For each action, write down a due date and a responsible. If the responsible is not present, write down who is responsible to give the information to this person.

Implementation: If you are planning to do measurements or tests in defined conditions write down not the measurements themselves but also all the boundary conditions. Give success criteria of tests or measurements in advance, too.
Even when you think that you did not get any concrete outcome from the review, you certainly have some outcome that can and should be made concrete in order to allow an elaborate progress of the project. This could be anything, even "soft" points like a common understanding of the problem or the awareness about some missing information. Write down a responsible and a due date for the communication of this outcome to the management or other departments in your company.
If not all the actions decided upon are absolutely crucial agree on a priority of the actions and also write it down.

Examples: Do not say, „Maybe further evaluation of the SW behaviour in extreme conditions necessary, " but
1. „Testing of SW version 12.3.4 on processor SmCa3.5. Stimulate temperature sensor input temp_1 with 50 and user interface input user_3 with 933 and measure display output for 3 hours while increasing the environment

temperature from -20 to +50 °C. Output has to be between 11 and 19.8. Responsible: Peter. Due date: Oct 31st 2009." or

2. „Peter and Heidi will meet in order to define the exact measurements. Responsible: Heidi. Due date: Sep 24th 2009." or

3. "Write down common understanding of temperature problem with SW version 12.3.4 on processor SmCa3.5 and present it to management in next team manager meeting. Responsible: Bob. Due date: Sep 10th 2009."

Acknowledgements

Many thanks to my sheperd Klaus Marquardt for his great support, his patience with me as a patterns beginner and his concrete feedback with many examples!

Also I would like to thank the members of EuroPLoP 2009 workshop C, namely Allan Kelly, Albena Antonova, Lise B. Hvatum and Wim Laurier. Their questions and feedback helped me a lot to improve my paper.

References

[1] Tom Gilb, Dorothy Graham, *Software Inspection*, Addison-Wesley, 1993

[2] Karol Frühauf, Jochen Ludewig, Helmut Sandmayr, *Software-Prüfung – Eine Anleitung zum Test und zur Inspektion*, vdf Hochschulverlag an der ETH Zürich, 5. Auflage, 2004

[3] Mark Prince, Andy Schneider, *Patterns for the End Game*, Europlop 2004, http://hillside.net/europlop/europlop2004/Papers/wwc/C3.pdf

[4] Jeffrey K. Liker, *The Toyota Way*, McGraw-Hill, 2004