TOWARDS IMPROVED REVIEW OF SOFTWARE DESIGNS*

Peter Freeman

November 8, 1974

Revised January 13, 1975

Technical Report #56

ABSTRACT

Review of designs is one aspect of the software creation process that has been neglected. Evaluation of the quality of a design, in particular, with respect to non-operational goals, is especially difficult. Design rationalization, a methodology currently under development, is proposed as a means of improving the reviewability of software designs. Primarily, it is a technique of explicitly recording design information that may be useful for reviews. In contrast to current review standards, it stresses the importance of documenting the reasoning used to arrive at a design. The technique is explained and a scenario illustrating its use is given. The paper concludes with a discussion of the strengths and weaknesses of design rationalization and a set of suggestions for implementing it in practice. The suggested approach is not guaranteed, but careful adaptation of it to particular situations should provide noticeable benefits.

This is a preprint of a paper to appear in Proceedings 1975 NCC.

*This work was supported by National Science Foundation Grant GJ-36414.

INTRODUCTION

A good deal of effort has been invested in recent years to improve both the form of programs and the processes used to create them [2,9,10]. As the payoffs from this work become apparent and more widespread [3], attention is turning to the form of designs and the processes used to create them [1,5].

One aspect of software design, reviewability, has received scant attention. In this paper, we want to stress the importance of making designs reviewable and suggest an operational technique for aiding in their review. Underlying our discussions is the principle (perhaps obvious) that designs which can be easily reviewed have a better chance of meeting the expectations of their purchasers and users.

A methodology being developed by the author, design rationalization , provides a means for making software designs more reviewable before they are actually implemented. The body of this paper develops this idea.

Before we begin, three important points must be made. First, what we are proposing here is an approach to the improvement of design practice. It is not an algorithm. It cannot be applied to a situation without some thought and study. It certainly cannot be guaranteed to work in all situations. If you demand instant success, then look elsewhere! But, if you are concerned with improving design practice in your organization, especially with respect to reviewability, then we believe the idea presented here merits your study and experimentation.

Second, we must stress that there are already well-specified procedures for reviewing designs, but that in spite of their good intentions, they fail in some important respects. Much of the procurement of software by the government is now controlled by standards (promulgated by the Defense Department and other parts of the government) that spell out elaborate review procedures that must be carried out during the design phase (for example, [11]). Additionally,

some organizations are experimenting with their own review standards aimed at improving the reviewability of software designs. (For example, good success has been informally reported by TRW at recent technical meetings with their usage of "unit development folders.") What these standards do not stress and what we consider essential to good review, is the recording of the reasoning behind design decisions, both local and global. It is this fact that our technique addresses most strongly and which we will stress in this paper.

The third point is that not all design situations are equal. We differentiate between discovery design and routine design. In the former, a great deal of creativity is required since the right structure (and even functions) for the software must be discovered during the course of the design. In the latter, the system being designed is similar to others which are well understood; thus routine design is more a process of choosing the right values for a set of parameters. Design of a program to prepare the payroll for an organization is clearly routine design. Design of a complex system to provide managers with real-time summary information automatically is discovery design, given our current understanding of such systems. In this paper we restrict ourselves to consideration of routine design situations.

We will describe and illustrate the design rationalization methodology and then show how it can be used to improve the reviewability of software designs. Because new methods are usually adopted slowly (and rightfully so), we close with some suggestions for experimentation with this technique.

DESIGN REVIEWS

Routine software designs are typically reviewed several times in different ways. Before we propose a way of improving design reviews, we want to consider some of their characteristics.

First, look at the range of review formats. An important part of the design process is a constant, but informal, review and iteration of the design by the designers themselves. When the preparation of a design is a large undertaking and/or is supported by a highly structured organization (such as the Federal government), formal design reviews are often specified (as in [11])at which people other than the designers determine if the proposed design is acceptable (by whatever standards have been set up). Finally, the ultimate user of the software will review the design informally through usage and sometimes formally in preparation for requesting changes or a new design.

Our concern in this paper is the reviewability of a design -- that is, the ease with which it can be compared to objectives. Designs cannot be executed directly as can programs, but it is still essential to compare them to desired criteria as early as possible in the development process.

While review of a design must necessarily mean different things to different people (depending on the methodology used, what is expected of the reviews, the stage of the process at which it is performed, and so on), let us be more explicit.

We see four possible components of any design review:

- checking for functional completeness;
- comparison of the design to operational goals and constraints;
- comparison of the design to non-operational goals and constraints;
- performance prediction.

"Operational" goals clearly and unambiguously spell out what is desired, while "non-operational" goals do not. For example,

Operational: "The system should provide a distinct error code for each error discovered."

Non-operational: "The system should handle errors cleanly."

It may be possible to characterize design reviews differently, but these four aspects capture most of what we see happening in the review of a software design.

The most prevalent question asked in a review is, "Will the system do what it is supposed to do?" The normal techniques of reading a design, perhaps aided by a structured walkthrough [8], will generally suffice for answering this question. Because a design is typically stated in functional terms, most of it speaks directly to the question of what the system will do. While existing techniques do permit review for the completeness of major functions, it is still difficult to ascertain from a design whether small or unwanted functions are present.

Likewise, for explicitly stated structural goals or constraints, existing design formats permit at least a passable review. For example, if certain data structures or interfaces are part of the design requirements, then it is usually possible to determine if these requirements have been met by inspecting the design.

It is when we come to the last two components of a design review -- comparison to non-operational requirements and performance prediction -- that the need for improvement becomes most apparent. Even though it may be possible to make some design goals more operational (that is, detailed and open to objective evaluation), reviewers will still be asked to evaluate designs with respect to non-operational goals. For instance, consider the following design goals:

"The system should be tolerant of user mistakes."

"Only state-of-the-art techniques should be used."

"Output formats should be neat and readable."

"The system should be maintainable."

Determining whether these goals have been met or not requires the reviewer to interpret or infer information from the design and to provide a good deal of external information. Typically, the information provided in a design document is not the right type and/or is in the wrong form to permit such goals to be evaluated directly (if at all). The reviewer usually must proceed unaided.

Finally, designs provide almost no help at all for performance prediction. Ad hoc comparisons of parts of the design to previous designs (for which performance is known) may provide some meaningful predictions of resource usage. But even if the parts can be identified from the design, determining which are critical to performance and what the interactions between parts will be is very difficult. The information needed is simply not present in most designs.

Cutting across all aspects of design reviewing is the need for knowing the reasoning behind decisions. Rarely, if ever, in current practice does design documentation record the alternatives that were considered and the reasons for rejecting some of them. Yet, this information can greatly aid the reviewer in understanding the design and in evaluating it against stated objectives.

This brief look at the nature of design reviews certainly does not exhaust what can be said about them, but it should set the stage for considering how to improve them.

DESIGN RATIONALIZATION

There is no argument that software designs should be more reviewable and that if they were the resulting implementations could be improved. This is especially true in the case of routine designs where the form of the result is pretty well known in advance.

We will outline below how design rationalization can be used to improve the reviewability of designs, especially in the area of recording the rationale for decisions.

The basis for design rationalization is the belief that designs can be improved by making them more rational. That is, design decisions should be based on logical reasoning, be supported by facts, and be recorded. The cornerstone of this technique is the explicit recording of design information in the form of design problems, alternative solutions, and the evaluations or arguments leading to the choice of a particular alternative.

The basic operation is the identification and recording of the information essential to a rational design. While variation in format is appropriate, the information shown in Figure 1 is fundamentally what goes into a rationalized design. The example shown there involves a single, rather low-level decision. In an actual rationalization we would record information pertaining to the entire design, both locally and globally.

If the information is collected and recorded as the design decisions are being made, we are doing a synthesis rationalization. If the information is primarily recorded after the design decisions are made, then we are doing an analysis rationalization. In either case, there are several important parameters: what features or decisions of the design shall be rationalized, how do we generate alternatives, what criteria shall be used for evaluating them, and on what basis should a decision be made.

Note that this methodology does not specify in what order decisions should be made. Neither does it spell out criteria for making decisions, except to specify that they should be made by considering alternatives and presenting evidence for and against each alternative. In this sense, design rationalization is more of a framework or forcing function within which particular decision strategies such as top-down or bottom-up can be used.

DESIGN PROBLEM 3: How should an error detected in a command string be handled?

ALTERNATIVE 3-1: Abort the program when an error is found.

EVALUATION 3-1: Easy to implement.
Provides the user very little information.
Wastes resources if the error occurs after much processing.

ALTERNATIVE 3-2: Ask the user to re-enter the command string.

EVALUATION 3-2: Must reset the state of the program.
Makes the system "softer" on the user.
Takes more processing time, even if no error encountered.
May prevent waste of resources if trivial error.

ALTERNATIVE 3-3: Try to correct the user's mistake.

EVALUATION 3-3: Maximally useful to user.
Substantial resources needed to try correction.
Interaction with user more complex (must specify correction and allow override).

DECISION 3: Alternative 3-2, because it provides a balance between our goal to make the system easy on the user and the constraint that it be fast.

Figure 1: Information Contained in a Typical Rationalization

We must stress that the important aspect of design rationalization is its insistence on the explicit capture and recording of design information including the reasoning used. Without this, we have nothing but motherhoods about the importance of making rational decisions -- which everyone already believes. With the explicit recording of information underlying decisions, however, we have a technique for increasing the rationality of designs.

With this brief introduction, let us look at how design rationalization can be used to improve the reviewability of designs.

A SCENARIO

The characteristics of a design review discussed above can be observed whenever a design is evaluated. For definiteness, though, let us focus here on the use of rationalization to improve formal reviews of routine designs.

Consider the following scenario:

1. Initial specifications are prepared. Assuming the specs are at a functional level, they are rationalized by providing explicit alternatives for critical specifications. Reasoning, based on facts is then spelled out for choosing a particular set of specifications.

2. Specifications review. The rationalizations permit the potential designers to understand more readily some of the specifications. They also permit those with funding responsibility to consider alternative forms of the system and to evaluate whether the system being specified is what is needed.

3. Initial Design. Once the specifications are finalized, an initial design is prepared, using the synthesis rationalization technique. Design decisions to be rationalized will include the overall organization of the system (control and data), choice of implementation language, choice of hardware, and other high-level decisions made at this stage. In addition, more detailed considerations of internal structure of the system may be documented by explicit lists of alternatives and evaluations.

4. Internal Review. When a design phase is completed, an internal review (such as the design walk throughs practiced by some organizations) and an analysis rationalization is performed by the designers and others in their immediate organization. This review may prompt changes to the design.

The system's features are more thoroughly explained, additional alternatives are provided, and the evaluations of alternatives are strengthened. Some evaluations can only be made on the basis of global considerations after the entire design has taken shape. For example, choice of a data structure may depend on its usage by several different modules. This internal review has the effect of catching some design errors in-house while at the same time improving the explicitness of the rationalization.

5. External review. The initial design, augmented by the rationalization, is thoroughly reviewed by whatever outside agency has been designated to monitor progress. Rejected choices are explicitly spelled out in the rationalization along with the reasoning used to reject them. This permits reviewers to assess better the quality of the design and its fit to the specifications. Rejected alternatives may be recognized by the reviewers as important to some of the goals even though the designers felt they were not.

6. Iteration and design refinement. After any design review, changes to the design may be needed. After these have been made the design process continues by refining the design (or extending it, depending on the approach being used). Using the techniques outlined in 3, 4, and 5, the design and review iterative cycle will continue until a complete design ready for implementation is obtained.

7. Implementation. It is rare that the implementors of a system can proceed without making any changes or additions to a design. Typically, many decisions (hopefully, low-level) concerning the structure of the programs being built must still be made. The rationalizations now play a role in a different form of design review. As the implementors seek to carry out the design, they must review it from the standpoint of understanding the intent of the designers when that is not clear and of making sure that decisions being made during implementation are not changing critical features of the design. The rationalization contains much of the reasoning information needed for this type of review.

8. Redesign. After a system is in use, more information is available on how well it fulfills its intended purpose. If the need for an improved system becomes clear then a major modification of the existing system or specification of a new system may become necessary. In this case, review of the design of the existing system to determine how it can be improved will be an important part of the design of the new system. This redesign process can profit from the rationalization by recovering rejected alternatives from the initial design.

This scenario illustrates the more important forms of formal design reviews often called for in the context of routine software design projects in large organizations. The use of design rationalization in these different review situations has been informally indicated, but the crux of the method --

the explicit capture of design information otherwise lost and its presentation in a form convenient for review and comparison to goals -- should be clear.

As a limited example of the usage of design rationalization, we performed a rather thorough analysis rationalization on a small system which had been designed and implemented as an improvement on an earlier system. The new system had several stated goals, including making it maintainable. The language used in the new system and some of the obvious features of the new design indicated that indeed this goal might have been achieved. The rationalization we produced, however, indicated that most of the effort in the redesign had been spent on local reorganizations of the system, with little thought given to overall control and code organization of the system. The careful analysis of the system that the rationalization supported convinced us that maintainability had been improved only marginally because of the lack of attention to overall structure. We were thus able to assess more accurately both the system and the techniques used to design it. This is illustrative of one type of benefit we would expect to reap from using design rationalization.

DISCUSSION

Our interest is in seeing design rationalization used to improve the the practice of software design. To facilitate this, we will discuss some of its advantages and disadvantages in this section.

Advantages

One advantage comes from helping a reviewer identify alternatives. If one is knowledgeable in the area of the design, then alternatives may spring to mind easily. However, many reviewers will not be experts and will have difficulty knowing what alternatives (if any) might have been chosen for the design. Even for the expert, generating alternatives is often not easy.

The value of a rationalization in this respect is twofold. First, actual alternatives that have been rejected will be readily available for the reviewer to consider. Information that has been generated during the course of the design will not have been lost, but will be available for the reviewer. Perhaps even more importantly, the rationalization can serve as a pump-primer to get the reviewer started to thinking about feasible alternatives for the question at hand.

We are all familiar with the effect of being presented with a problem situation and of seeing at first only one solution. Unless one is familiar with the content area and has thought about the problem previously, it takes some effort to seek out alternatives. However, if someone suggests an alternative, even if it is not a good one, then we often can come up with additional suggestions much more easily. This is similar to one of the techniques suggested by De Bono [4] for facilitating lateral thinking -- that is, of finding new ways of looking at an old situation.

Fundamentally, a reviewer is asked to certify that the decisions made by the designer are good decisions with respect to the goals and constraints of the design task. If the reviewer knows only the results of the designer's reasoning and not the steps by which the decisions were made, then the biases and knowledge limitations of the reviewer may seriously effect his or her judgments as to the quality of the design decisions. If the design is supported by explicit information in the form of a rationalization, however, then the reviewer can assess more easily the factual evidence and logical reasoning used by the designer.

This situation has an analogy in mathematics. If one is presented with a theorem, the truth or falsity of it may not be immediately evident and we may or may not be prepared to accept it as true. Given a step-by-step proof of the theorem, however, we can convince ourselves not only of its truth but also of why it is true (in terms of axioms and reasoning).

While the evaluations in a design rationalization are nowhere near as orderly as a mathematical proof, they do present the reasoning that has been used so that others can decide for themselves whether that reasoning is complete and valid. In addition, where the reasoning used to make a decision involves assumptions, exposing this reasoning will permit the reviewers to discover and assess the validity of the assumptions (since many of them may be related to the user environment which the reviewer knows more about anyway).

The advantages discussed here have touched on what we believe to be the basic advantage of design rationalization for improving reviewability: It forces the explicit recording of decision reasoning information which is otherwise lost.

Disadvantages

It should be clear that producing a rationalized design will, in general, take more effort than producing an unrationalized design. At a minimum, the effort needed to record alternatives and evaluations, even if otherwise generated, is added effort. However, we have typically found that generating the explicit lists of alternatives and evaluations also requires a good deal of effort. Finding meaningful evaluations, that realte the decision under consideration to the goals and constraints of the problem, is often difficult in the absence of an underlying theoretical basis or quantitative evaluation technique. The advantages of design rationalization (both for reviewability and quality of a design) must be weighed against this added cost.

At present we cannot offer explicit suggestions for choosing the design problems/features to rationalize nor sure- fire methods of generating alternatives and evaluations. It is clear that one cannot rationalize every single decision in a program of any significant size. Further, important decisions in one design may have no major role in another. Choosing the important ones (those for which the choice of a solution has some definite effect on the resulting design or its use) is difficult.

A more subtle problem that may not be immediately evident concerns the level at which decisions are made. The thrust of design rationalization as we have presented it above is to make decisions at a local level. Basically, the question asked is, "What is the best alternative, and why, for this particular decision?" This leads to local optimization, which in many cases will not be optimal. That is, sometimes we must make decisions taking into account the alternatives for other decisions which have not yet been resolved. It is to help alleviate this concentration on the local context that we have suggested in the scenario above that some rationalization be done after the

initial design is completed.

Research

We are continuing our investigations of design rationalization both to provide additional evidence of its advantages and to find ways of reducing its disadvantages. Included in this work are some informal investigations of design situations in which some designers use rationalization and others do not, development of techniques to make easier the choice of problems and generation of alternatives and evaluations, and the construction of tools to help in recording the information. These studies are described in [6] and [7] and other working papers available from the author.

SOME SUGGESTIONS FOR USING DESIGN RATIONALIZATION

Any methodology not based on formal techniques is open to interpretation by those using it. Such interpretations are, in fact, required in most cases to make methodology useful in the context of a particular organization or a particular type of task.

Thus, we understand that design rationalization must be adapted to your particular organizational task context. While it is difficult to predict the difficulties you will encounter, our limited experience with helping others use it does suggest some guidelines.

Remembering that we are concerned here with formal reviews of routine designs, we suggest the following:

1. Choose a small, but realistic design problem on which to try out the technique. Make sure the design is of a system of which the designer has some knowledge.

2. Use a design project that is "real" (not done just for experimentation).

3. Make sure sufficient time and resources are allocated to the project so that the designers are not under pressure.

4. Use at least two designers (but probably not more than three) so that they can work as a group when generating alternatives and evaluations.

5. Use your normal design techniques augmented by the use of rationalizations as suggested in the scenario above.

6. Carefully choose some criteria by which you can judge whether the rationalization assists in design review. Some suggestions are: number of design flaws discovered relative to similar projects, level of detail of design flaws discovered, perceived ease of review by reviewers, time taken to review design documents.

7. Maintain careful observations of the use of rationalization to permit later analysis of the trial.

8. Assess the trial when completed. If rationalization seems to help in your situation, even a little, try to find ways to improve the technique for your situation.

9. Try it again.

These suggestions can be boiled down to a simple statement: Approach the use of design rationalization from the standpoint of an experimental fitting of an idea to your situation and expect to make changes.

CONCLUSION

Development of techniques for the review of software designs has been largely neglected. We have described a methodology, design rationalization, which has characteristics that will help improve the reviewability of designs. We have given a scenario for its usage and discussed some of its advantages and disadvantages. Suggested guidelines for trying it out were given.

We have stressed that the strength of design rationalization lies in its forcing explicit recording of design information, especially that which explains the reasons behind features of the design. The existence of this information in a form that permits independent review of design choices and the reasoning leading up to them should assist in most situations.

We have not spelled out an explicit technique for one to follow. Rather, we have described an idea, assessed its use for improving the current practice of design review, and suggested ways in which it can be adapted to varied organizational settings. As with much of software engineering today, the application of this idea in large-scale situations must ultimately be carried out by those with software creation problems to solve.

While our research continues into the ramifications of this idea and the techniques for using it, others can profit from trying it in their contexts. We recognize the difficulty of changing one's patterns of doing something and the difficulties in forcing oneself into the discipline of design rationalization. Yet, only through trial and error usage of this and other proposed methodologies can we gradually develop the tools necessary for the routine design of large and important classes of software.

REFERENCES

1. Brown, R.R. "1974 Lake Arrowhead Workshop on Structured Programming", Computer, October, 1974, pp. 61-63.

2. Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R. Structured Programming, Academic Press, 1973.

3. Datamation. Special issue on structured programming, December 1973.

4. De Bono, Edward. New Think, Basic Books, 1972.

5. Freeman, Peter. "Automating Software Design", Computer, April 1974.

6. Freeman, Peter. "Reliable Software Through Rational Design", ICS Technical Report #55, University of California, Irvine, October 1974.

7. Freeman, Peter. "Design Rationalization", ICS Technical Report #57, University of California, Irvine, November 1974.

8. IBM. "Structured Walkthroughs", training brochure.

9. Mills, H.D. "Top Down Programming in Large Systems", in Debugging Techniques in Large Systems, R. Rustin (ed.), Prentice-Hall, 1971.

10. Parnas, D.L. "On the Criteria to the Used in Decomposing Systems into Modules", Comm. ACM, December 1972.

11. U.S. Air Force MIL-STD 1521, "Technical Reviews and Audits for Systems Engineering and Computer Programming", available from National Technical Information Service, Springfield, VA., September 1972.