MANAGING THE DEVELOPMENT OF RELIABLE SOFTWARE

R.D. Williams

TRW Systems
Redondo Beach, California 90278

Keywords and phrases. Reliable software, software reliability, production methodology, Site Defense, real-time software, requirements engineering, preliminary design, detailed design, implementation, validation testing, requirements traceability, process construction, tools, incremental development, Unit Development Folder, independent test group, discrepancy reporting.

Abstract. This paper discusses the serious need for improved management methods and software development technology to support the production of truly reliable software and presents experience gained on TRW's Site Defense Program in addressing that need.

The basic theme is simply this: We have attempted to achieve a better understanding of how to specify, design, build and validate reliable software; we have translated that understanding into an integrated and exceptionally disciplined software development methodology; we have used, evaluated and refined the methodology enough to recognize and report substantial early success and continuing progress toward that goal.

Introduction

The subject of this conference is reliable software. Participation in it is due, presumably, to a desire to seek and find a better understanding of both how to get reliable software and how to know when it is. In recent years there have been many conferences, symposia, and workshops which have focused on software reliability and there will be many more in years to come.

The term "software reliability" is relatively young compared to the thirty or so years that we've been developing software. The need to develop highly reliable software outdates the term itself by quite a few years as does the corresponding search for an improved production methodology to do the job. If that search had led to definition and universal application of an ultimate reliability-oriented software production methodology, then we probably wouldn't be here as avidly discussing the subject as we are now. But that's not to say that there has been no progress in that direction. In fact, at TRW, where the search has been intense and continuous over the years, a great deal of progress has been made, a lot has been learned, and we can say conservatively that we've come a long way.

As the project manager for TRW's 100 million dollar Site Defense software development program, I am as concerned as anyone about the production of reliable software. Given the need for highly sophisticated, critical functions to be performed by a large, real-time software system, it was clearly appropriate to bring together TRW's latest and best reliability oriented production techniques to form an integrated Site Defense software methodology. We have done so and have continued to refine and supplement the methodology with additional methods which directly support the building-in of reliability as the development effort proceeds.

The following discussion highlights a number of techniques and project activities which have proven to be especially valuable in our efforts to develop truly reliable Site Defense software. We have attempted to take positive steps to implement and faithfully apply a reliability-oriented development methodology; I hope to report enough of the details of our experience (what we've done and what we've learned) to show that there is very real hope for those who need now to cope seriously with the software reliability issue.

The site defense approach

The Site Defense approach to software development has been built upon the rather conventional stepwise production process illustrated in Figure 1.

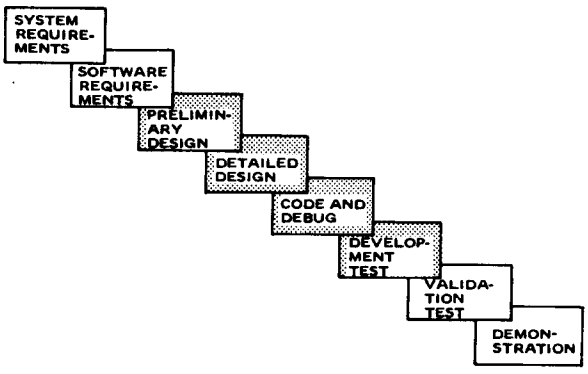

Figure 1. Software Development Approach

The 8-step process, as shown, is admittedly a gross simplification of what goes on in carrying an actual, large-scale, real-time software development effort to completion. It does, however, contain the essential ingredients and thus provides a frame of reference upon which identification and discussion of unique features of the Site Defense development methodology can be based. In combination the unique features span the complete process, however, most of this discussion is devoted to those things we have done and demonstrated to be an especially beneficial adjunct to the shaded steps (i.e., Preliminary Design, Detailed Design, Code and Debug, and Development Test).

In practice the uniqueness of the Site Defense software development methodology takes a number of different forms when viewed in a total project context. First, as previously mentioned, there are many special additional functions which supplement the basic process at key points to directly support the finding and fixing of potential problems before they become real ones. Another form of reliability-oriented feature consists of the differing emphasis or proportional importance attributed to the distinct steps of the development process. Within the Site Defense methodology, an unusually large share of attention is given to the detailed specification and unambiguous interpretation of software requirements and the development of a complete and consistent preliminary design of the total software system. The goal is to produce a preliminary design which contains:

1) major software functions which either directly correspond to or can be easily traced to explicit software requirements,

2) a complete picture of the overall structure of the software system,

3) enough detail regarding the allocation of functional processing requirements to designated software elements to support a thorough and credible demonstration of design feasibility and validity.

Finally, the evolution of the Site Defense development methodology has been influenced by a number of guidelines, lessons learned, rules of thumb, principles of good management and other sage advice derived from both pleasant and unpleasant experiences associated with past software development activities. In [1], Royce identifies what he calls features that must be added to the basic development approach to eliminate known development risks. The features are:

1) Complete program design before analysis and coding begins

2) Documentation must be current and complete

3) Do the job twice if possible

4) Testing must be planned, controlled and monitored, and

5) Involve the customer.

It's not necessary to elaborate further upon these important principles except to state that they have all been refined and applied in some form and, in that fashion, have come to be integral to our development approach. The specific effect of these principles and others in contributing to the successful production of reliable, safe, sane, secure and maintainable Site Defense software will become more apparent from the following discussion.

Specifics – what have we done and what have we learned?

Requirements engineering. There is nothing, absolutely nothing, that can cause a more devastating outcome of a development activity than an incomplete and/or incorrect knowledge and corresponding specification of system and software requirements. If the customer does not have a good understanding of what needs to be done or if the contractor can't figure out what the customer wants, then there is good reason to believe that serious trouble lies ahead.

On the Site Defense program, we have found several measures which cope well with this potentially serious problem. For one, we have defined and carried out a continuing series of reviews for the specific purpose of engendering a mutual understanding of both the requirements (as stated) and the actual customer intent (if different). The reviews have been very successful in that they have surfaced problems which might have otherwise gone unnoticed and served as a medium through which the problems could be resolved in a timely and mutually agreeable fashion. In addition, we have benefited from the use of an innovative and extremely powerful technique called Engagement Logic. Very briefly, Engagement Logic gives us a way to express system or software requirements by way of illustrating the implied connectivity of required high-level functions which, in combination, directly relate to overall information processing requirements. Engagement Logic diagrams have so far provided needed insight into potential requirements inconsistancies and incompleteness and have prompted corresponding effort to explore and eliminate the problems before they became just that.

Despite having introduced unprecedented rigor into the task of specifying and reaching mutual agreement on unambiguous requirements, we fully appreciate the need for even more rigor and a comprehensive technology to guide and control the requirements specification effort. Under contract to the Ballistic Missile Defense Advanced Technology Center, TRW is in the process of developing just such a technology. We have, in fact, now completed a large part of the work which includes identification of the appropriate contents and format for both system and software requirements specifications, development of a requirements specification language, design and development of selected tools to support both generation and validation of requirements, and definition of a total requirements engineering methodology which incorporates all of the above.

It is difficult, if not impossible to estimate the number and severity of problems already avoided through systematic application of a disciplined approach to requirements engineering on the Site Defense program. Above all, however, we are encouraged by our successes to search still harder for even better answers than we now have. The prospect

of finding and using those better answers, and with it the likelihood of producing even more responsive and reliable software than before possible, are getting better and better all the time.

Software design. No matter how you slice it, the creation of a highly sophisticated, yet implementable design solution which will clearly satisfy demanding design, functional processing and performance requirements is far from an easy task. It would be foolish, in fact dishonest, to claim or even imply that we have made some fantastic breakthrough that has made Site Defense software design easy. On the other hand, we have taken steps to identify peripheral problems which make the job harder than it has to be and taken further steps to eliminate the problems or reduce their impact on the design process.

One of the main steps has been to recognize the importance of the completeness and consistency of the preliminary design. This recognition directly resulted in a development approach which placed unprecedented emphasis on the preliminary design effort. It also led to necessary management decisions and action to ensure the timely availability of a small team of senior system/software designers. The primary tasks of the senior design team included:

1) the top down development of an overall software system structure providing visible evidence of traceability to requirements at the major software function level and thus providing a constructive approach to handling the "program complexity" problem discussed by Mills [2],

2) the formulation and documentation of explicit design guidelines to be followed by all designers during subsequent detailed design effort, and

3) early, detailed definition and documentation of the data base structure, contents, and access rules.

Of the design guidelines developed by the senior design team, those most directly related to specific reliability concerns include:

1) Clearly identify those requirements which can best be satisfied through implementation of operating system functions with special attention to requirements which are common to several application programs.

2) Think functional modularity!

3) Extend top down concepts into the detailed design and structuring of application modules. For every application task, develop a supervisory task control routine to guarantee sufficient visibility into the functional capability provided and localize and reduce the impact of possible requirements modification.

4) Review, analyze, and validate at each step. In particular, perform necessary updates to the software system Engagement Logic process control diagram to ensure that new design is consistent with the established baseline and that functions are indeed traceable to specified requirements.

5) Identify, develop and use design support tools and proven techniques such as:

   - the software system N^2 - chart portraying latest information on the data needs of each module, inter-module data dependency, and the availability (or unavailability) of data elements during process execution,
   - automated data base and N^2 - chart update tools (eliminating one source of troublesome errors which frequent equivalent manual efforts)
   - an executable dummy process in sufficient detail to demonstrate actual availability of needed data and ensure consistent interpretation and usage of data across all software modules.

It is possible to briefly summarize both our overall approach to Site Defense software design and the primary accomplishments to date as follows:

1) We looked hard at past TRW experience and the experiences of others to determine the necessary content and appropriate level of detail for preliminary and detailed design specifications.

2) We took all required actions (i.e., formation of a senior design team, formulation of explicit design guidelines, and timely development and application of proven design methodology and support tools) to make sure that the right resources were available and applied to the right problems at the right time.

3) We succeeded in producing a completely credible preliminary design which in turn forced the establishment of a firm development baseline providing for:

   - solid software design feedback to customer/system designers,
   - a basic software structure for subsequent detailed design,
   - visible capability of the software (process) structure to accommodate changes,
   - detailed determination of timing characteristics in a realistic environment under load through use of both low and high fidelity simulations,
   - documented traceability between requirements and design.

Implementation. Nearly all large software projects experience a relatively dormant period which begins during the final throes of detailed design and

continues until intelligible results are produced by an executing computer program. The activity spanning this period of time, often called implementation, has received more attention (with regard to speeding it up and doing it better) than any other phase of the typical development process.

In particular, the need to:

1) reduce the normal wait time before one could "hear the process talk",

2) plan the implementation effort in detail, and

3) maintain continuous and instant visibility into implementation status

pointed to the development and enforced use of specialized implementation methods and tools as well as an integrated and well-defined management approach to planning, monitoring and controlling the implementation job.

The first need (above) was directly addressed in several ways. One of the earliest efforts undertaken on the Site Defense Program was to define the Process Construction approach to real-time software process development. This effort also included development of automated support capability in the Process Construction Program (PCP). PCP provides support to three major development functions of data base, system component, and system (process) building as illustrated in Figure 2.

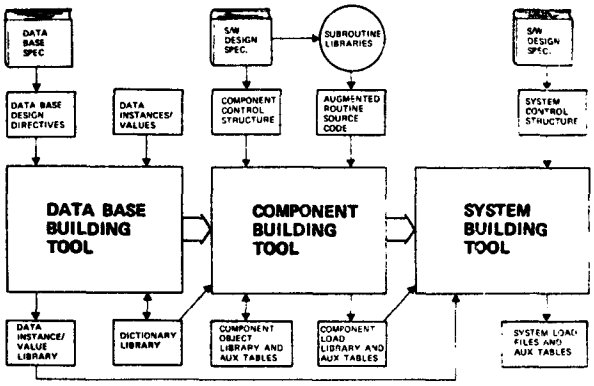

Figure 2. The Process Construction Approach

Other features of the Site Defense approach addressed the need to "hear the process talk as soon as possible" and the other needs as well. The most prominent of these features are:

1) an incremental approach to detailed design, code and test,

2) continuation of top down concepts in conjunction with the incremental approach through identification and development of the dummy process, loops and builds (illustrated in Figure 3),

3) early creation and continued use of the Unit Development Folder (UDF) to force early attention to process, task, and routine documentation and support detailed planning and monitoring of project effort and status.

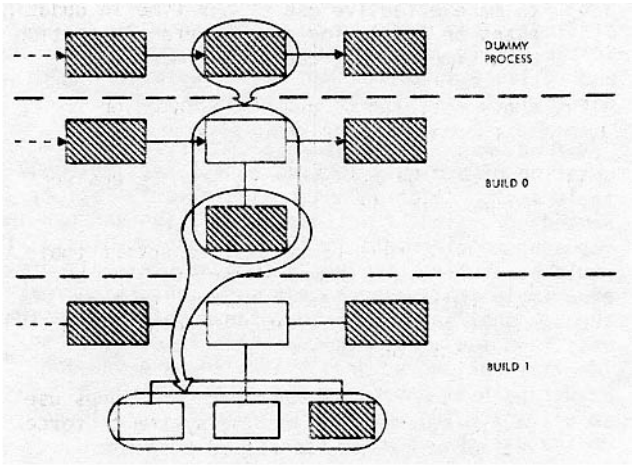

Figure 3. Loop 3 Task Development

In practice, 1 and 2 (above) are possible because of the earlier development of the overall software structure during preliminary design. Within that structure it is possible to identify selected portions of the software (e.g., processing threads involving critical functions) which can be developed and tested as independent increments. With care, these increments (or loops) can be chosen to provide a complete increment of system capability and, thereby, substantially reduce the likelihood of last-minute surprises.

The UDF is much more than a document. It does serve as a collection point for all pertinent development and test information and systematic updating of UDF contents ensures that documentation is actually done as part of the development activity. A mandatory part of the UDF is a cover sheet which summarizes the current development status of each program unit. Use of the cover sheet to accomplish unit level planning and control causes us to:

1) account for all of the required efforts that go into production of a unit of code,

2) address real functional activities for which designers, programmers and testors can make responsible estimates of time and effort to complete.

The UDF has become a tool to help the developer see impending difficulty and take necessary steps to stay out of trouble. Perhaps most important of all, the use of the UDF has helped us:

1) to obtain meaningful estimates through direct involvement of project personnel in scheduling their own work,

2) to accomplish continuous monitoring and accurate reporting,

3) to avoid the proliferation of phantom problems (of the kind discussed by Brooks [3] arising from poor estimates), and

4) to make effective use of our time in updating plans or initiating other corrective action at a time when it can do the most good.

The above features of our implementation approach, especially when coupled with:

1) the early establishment of comprehensive programming standards,

2) the development of automated support tools (e.g., PCP and Code Auditor) for use by developers, quality assurance and test personnel to ensure compliance with the standards, and

3) the design, development and continuous use of a discrepancy reporting system to force the close-out of identified problems .....

all these things should be ample evidence that we are indeed very serious about the job and intent on doing it the best way we know.

Validation testing. Given all the good things above describing our disciplined approach to requirements engineering, design and implementation, one might logically expect to experience fewer problems (than usual) during validation testing. We have become concerned, as have others [4, 5], about the general tendency of validation testing to consume ever greater shares of development time and cost. We have attempted to push the problem-finding and fixing effort further and further toward the front end of the Site Defense development process in hopes of reversing that trend.

The validation testing activity is not yet in full swing on the Site Defense Program. Enough has been done, however, to indicate fairly conclusively that many classes of problems either never occur or are cut off at the pass and, therefore, aren't around later to perturb the planned validation testing effort.

The Site Defense validation testing methodology can be briefly characterized as follows:

1) The primary objective is to verify that the software satisfies the requirements levied upon it.

2) The effort is performed by an independent test group.

3) All software to be validated must meet specified standards prior to turnover to the test group.

4) Software is placed under formal configuration control before validation testing can proceed and all subsequent changes are subject to documented procedures.

5) The test group efforts are audited by Product Assurance for sufficiency and compliance with test plans, procedures and specifications.

6) All errors are documented in discrepancy reports, deficiencies are analyzed and corrected by designers, and Product Assurance audits everybody to ensure timely and satisfactory problem resolution.

7) A number of test tools have been developed and are regularly used to supplement and improve upon the testor's efforts by completely handling otherwise tedious, error-prone, and difficult testing tasks [5].

The statement made earlier about making software design "easy" is equally true with regard to validation testing. Testing is an activity which requires lots of early attention and deep thought, much planning, preparation, analysis and more deep thought. That adds up to hard work. What we have found is that there are good ways to reduce the problem load on the testor and to eliminate the need for manual performance of tedious, often repetitive and certainly boring tasks. This has made it possible for the independent testing group to concentrate on real testing goals and on the complex activities and support software needed to meet those goals in contributing to the cost-effective production of a highly reliable end product.

Conclusions

The preceding discussion is full of conclusions, and there is little reason to repeat or rephrase them here. We've learned a great deal more about the exacting nature of reliable software development than could possibly be presented in this paper. We admittedly are proud of what we've done and what we've learned.

Many of the techniques and procedures are clearly not unique to TRW. The particular combination of the techniques into a working, highly-disciplined and demonstrably effective development methodology, however, is clearly unique as is our level of committment to continued application, learning, and refinement of the methodology for even better results. So far we've found that the goal of reliable software is worthy of our very best efforts, and we have more than enough reason to give it just that.

Finally, it should be noted that TRW is not alone. We are not alone in appreciating the increasingly severe requirement for production of highly reliable software [6,7]. We are not alone in understanding that much progress is possible through concentrated attention on the front end of the production process [8], as necessary to find and use improved ways to specify reliability requirements [9], and constructive methods for designing and building it in [10,11,12]. We are not alone in efforts to more fully understand what testing can and must contribute to the reliability issue [13], nor are we alone in the search for significantly improved testing technology to match the need [14,15]. We are, hopefully not alone in our willingness to openly discuss what we've done, what we've learned and what we think must be done next in continuing to advance toward a definitive and effective reliability oriented software production methodology.

Acknowledgements

This paper reflects not only my personal views but the combined experience, insight, opinion and outlook of many individuals who have worked for and with me in creation of a successful Site Defense software development methodology. I wish to

express my appreciation for their cooperation and numerous contributions. I owe special thanks to J.R. Brown for collection, review and documentation of the experiences, for evaluation of the impact of selected techniques on reliable software production, and for much direct support in the preparation of the paper.

References

1. Royce, W.W., "Managing the Development of Large Software Systems: Concepts and Techniques," TRW Software Series SS-70-01, August 1970.

2. Mills, H.D., "The Complexity of Programs," in Program Test Methods, ed., W.C. Hetzel, Prentice-Hall, 1973.

3. Brooks, F.P., Jr., "The Mythical Man-Month," Datamation, December 1974, pp. 44-52.

4. Shooman, M.L. and Bolsky, M.I., "Software Errors: Types, Distribution, Test and Correction Times," Proceedings of 1975 International Conference on Reliable Software, April 1975.

5. Brown, J.R., "A Case for Software Test Tools," Proceedings of the TRW Symposium on Reliable, Cost-Effective, Secure Software, TRW Software Series SS-74-14, March 1974.

6. Israel, D.R., "An Overview of the Upgraded Third Generation Air Traffic Control System," Proceedings of 1974 IEEE Electronics and Aerospace Systems Convention, Cat. No. 74 CHO 883-1 AES, October 7-9, 1974, pp. 244-249.

7. Brown, J.R. "Improving Quality and Reducing Cost of Aeronautical Systems Software through Use of Automated Tools," Proceedings of the Air Force Aeronautical Systems Software Workshop, Dayton, Ohio, April 1974.

8. Boehm, B.W., McLean, R.K., Sewell, J. and Urfrig, D.B., "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," Proceedings of 1975 International Conference on Reliable Software, April 1975.

9. Vick, C.R., "Specification for Reliable Software," Presented to 1974 IEEE Electronics and Aerospace Systems Convention, October 7-9, 1974.

10. Carpenter, L.C. and Tripp, L.L., "Software Design Validation Tool," Proceedings of 1975 International Conference on Reliable Software, April 1975.

11. Liskov, B.H., "A Design Methodology for Reliable Software Systems," Presented at 1972 Fall Joint Computer Conference, December 5-7, 1972, AFIPS Conference Proceedings, Volume 41, 1972 FJCC, pp 191-199.

12. Horowitz, E., et al., "Practical Strategies for Developing Large Software Systems," Addison Wesley Pub. Co., Reading, Mass., May 1975.

13. Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R., "Structured Programming," Academic Press, London and New York, June 1972, p. 6.

14. Stucki, L.G., "New Assertion Concepts for Self-Metric Software Validation," Proceedings of 1975 International Conference on Reliable Software, April 1975.

15. Brown, J.R. and Lipow, M., "Testing for Software Reliability," Proceedings of 1975 International Conference on Reliable Software, April 1975.