Considerations for the Design of Quality Software

Jan Stambaugh
Hewlett-Packard Company
Business Computer Group

The design of software, like the design of hardware, is a science and not an art. Computer programming is a discipline and, as such, must be disciplined. There are specific procedures and considerations which can help to ensure reliable, high quality, software products.

What is meant by quality? Quality is the basic characteristic or property of something. A particular property could be physical, such as weight, size, and color, or chemical, such as composition. For example, the properties of a batch of chocolate fudge might be stated in terms of the hardness, the sweetness, the size and shape of the pieces, and so on. Any element that can be used to define the nature of a product can be called a quality characteristic.

How then can we define software quality? Software quality can be looked at in two ways: first, the quality of the design itself and, second, the quality of the conformance to the design. The first classification deals with the degree of excellence of the ideas which define the product design. The second classification relates to the degree of excellence to which the product conforms to the design specifications.

The goal in software development is to create software that performs reliably, meets user requirements, and does nothing that it is not supposed to do. However, there is currently no standard definition of what qualities should be considered in developing software. There is no standard means for measuring quality quantitatively. Because the production of software is finally making the transition from an art to an engineering discipline, quality is now being given more objective and less subjective consideration. While the current state-of-the-art in software design imposes specific limitations on our ability to automatically measure quality, software researchers and developers are beginning to find ways to evaluate the quality of software quantitatively.

How can quality be designed into software? There are three primary considerations. The first is the establishment and adherence to standards for the use of a (1) STRUCTURED SYSTEMS DEVELOPMENT METHODOLOGY.

At Hewlett-Packard, software products developed by the R&D labs follow a product lifecycle plan. Many of the elements of this plan are appropriate and relevant to the design of applications software by our users.

The software product lifecycle plan stresses the importance of scheduling by first defining all the tasks which need to be done. It suggests that software projects are frequently late because the amount of work required to complete the entire project is not made obvious from the start. A software project must be divided into as many identifiable operations as possible. The product lifecycle helps to simplify scheduling by identifying the milestones which are common to all software projects and by explaining how to determine other factors which may be unique to a particular project.

Documentation is necessary for the long term success of any software effort. The product lifecycle proposes a standard for documentation which defines what needs to be recorded as well as when it should be. It also addresses the archiving of these documents so that they can be easily referenced at a later date.

The project review cycle advocated by the lifecycle plan defines formalized sign-offs during the progression of a product's development from one phase to the next. The review cycle provides a check and balance function to ensure that what is being developed is in fact what will be needed.

The lifecycle plan describes two types of project classifications for every software project. These classifications provide guidelines to the project management team in establishing priorities. The first classification deals with the value of the project relative to all concurrent efforts in each of the functionally responsible areas. The second classification is a development category. Each project is tagged as an enhancement, a new product, or the conversion of an existing product. This second classification conveys the extent to which previous projects may reduce the development effort.

The software product lifecycle stresses an iterative design philosophy. The design of a product is continually refined throughout the product's pre-release cycle. Rather than finalizing the external specifications and then beginning the internal design, the software product lifecycle proposes that both be outlined early on and refined as the project progresses. This allows the project team to maintain a global perspective of both sides of the development effort.

Project review policies and procedures are defined explicitly in the product lifecycle plan. A lab review team, chosen by lab management, is responsible for the evaluation of one specific project from start to finish. The product team members are chosen from marketing, manufacturing, the R&D lab, and product assurance and the responsibilities of each member are defined in detail.

Each project utilizes the software development network, a generalized PERT-like scheduling tool designed to aid in project planning and control. It provides an overview of the milestones to be met and an indication of their relative timing.

Each software project goes through five phases: investigation, design, implementation, testing, and release. During the investigation phase, a proposal is developed which describes a possible area for a software contribution, suggests a software project team, and sets a date for reviewing the research findings. The investigation report describes the results of the research and identifies pertinent issues which have a bearing on the decision of whether to continue with the development of a product. A product datasheet is generated to provide a quick overview of the product's key points. During the investigation phase sign-off review, the fact that a complete investigation has been conducted by the product team is verified. A decision can now be made as to whether or not to proceed to the design phase.

The design phase of product development is where the product is defined in detail. Two major documents are produced during this phase: the design outline and the external specifications. The design outline is a first draft description of the internal structure necessary to implement the product. It establishes the basic system modules and identifies key shared data structures and tables. It also establishes a plan for the detailed design of the product. The external specifications describe the functions of a product and how to use it. They provide the basis for the internal design of the product. The external specifications include hardware requirements and restrictions, software requirements and restrictions, user documentation requirements, detailed functional specifications, individual function descriptions, the user interface, compatibility specifications, security specifications, installation instructions, performance predictions, reliability/recoverability specifications, special capabilities and features, and error messages, meanings, and actions. A resource and schedule summary is created at this stage, outlining the financial cost of the project at each major milestone and and updating previous estimates for project completion. The design phase sign-off review provides a formal review of the external characteristics of the proposed product to ensure that the product team is in agreement that what is being proposed is in fact the appropriate solution to the problem being addressed.

The implementation phase involves the creation of the internal design document. This document describes in detail the algorithms and the data structures to be used in implementing the product. It serves as the internal documentation for the product throughout the remainder of its lifecycle. During the implementation phase, test sites are selected. There are three types of test sites: Alpha, Beta, and foreign language. An Alpha site is internal to HP; a Beta site is typically external; a foreign site may be either, depending on the scope of the project. For a product to go to Alpha test, it must be functionally complete and have very few known bugs. For a product to go to Beta test, there must be no known bugs that would seriously impede the user, and the preliminary documentation must be complete. Just prior to the development phase sign-off review, the product team is responsible for making a presentation to the lab which serves as a brief introduction to the use of the product. Once again the resources and schedule summary is updated to outline the financial cost of the project at this stage and to establish updated estimates for project completion.

A product cannot be developed and then tested. Consequently, testing is not really a phase but an integral part of the development and release of a high quality software product. The goal of testing is to uncover errors and deficiencies at the earliest possible moment, thus eliminating the possibility of fatal surprises. The majority of the test effort is aimed at ensuring that the end product fulfills the original specifications and that the particular implementation of the product is well executed. Many different testing techniques and processes are used. A code inspection is a set of procedures and error-detection techniques for group code reading. The general procedure involves the distribution of a program listing and related design specifications to participants several days in advance of the inspection session. The programmer narrates, statement by statement, the logic of the program. Questions are raised to determine if errors exist. The program is analyzed with respect to a checklist of historically common programming errors. The errors identified are also analyzed and used to refine the error checklist to improve the effectiveness of its future use.

The structured walkthrough, like the inspection, is a set of procedures and error-detection techniques for group code reading. Rather than reading the program or using error checklists, the walkthrough participants "play computer." A person who has been designated as the tester comes to the meeting armed with a small set of paper test cases for the program or module. During the meeting, each test case is mentally executed.

Module testing, or unit testing, is a process of testing the individual subprograms, subroutines, or procedures in a program.

Incremental testing or integration is a method of combining the next module to be tested with the set of previously tested modules before it is tested. Incremental testing has two strategies: top-down testing and bottom-up testing. The top-down strategy starts with the initial module in the program. The rule for the next module to be eligible as the next module to be tested is that at least one of the module's calling modules must have been previously tested. The bottom-up strategy starts with the terminal modules in the program, the modules which do not call other modules. Here the rule for a module to be eligible as the next module to be tested is that all of the modules it calls must have been tested previously. Top-down testing requires the generation of stub modules and bottom-up testing the generation of driver modules.

Function testing is the process of attempting to uncover discrepancies between the program or system and its external specifications. It is not intended to check out the interactions between functions, but rather the functions themselves.

System testing is a process used to compare the system or program to its original objectives. It is a set of tests to verify that all components work together harmoniously. System testing includes the following: facility testing, volume testing, stress testing, usability testing, security testing, performance testing, storage testing, configuration testing, compatability/conversion testing, installability testing, reliability testing, recovery testing, serviceability testing, documentation testing, and procedure testing.

Acceptance testing is the process of comparing the program to its initial requirements and the current needs of its end users. This testing is accomplished by the test sites.

Installation testing takes place as a means of finding installation errrors. The test cases check to ensure that a compatible set of options has been selected, that all parts of the system exist, that all files have been created and contain the necessary contents, and that the hardware configuration is appropriate.

The testplan is a crucial part of the testing process. It outlines the types of test which will be used by the project team and specifies any non-standard tests, as well as the frequency to be used for repetitive operations such as walkthroughs. It should define a set of tests which will be sufficient to guarantee the quality of the finished product upon release from the lab. The testplan is developed as an integral part of the design and implementation phases, and not as an afterthought.

The automated test specification documents individual test programs, data sets, and procedures. It describes the purpose of each test as well as providing some detail about the internal workings.

The release phase involves reliability certification, performance specification and tuning, and turning control of the product over to manufacturing. Up until this phase, performance has been an issue of prediction and calculation. Now it is an issue of measurement. What the end user will see must be quantified, and the internal quantities which might affect these tangible values must be identified. The manufacturing release sign-off review establishes the product's completion and readiness for sale and distribution, and signals its release from the R&D lab and its entry into the maintenance phase of the product lifecycle. Six months after the product has been released to the field, a post release review meeting is called for the purpose of reflecting on the acceptability of the product in the marketplace. All members of the product team are required to attend.

The second consideration for the design of quality software is the use of (2) QUALITY-ENHANCING TOOLS AND TECHNIQUES. Some of the tools for enhancing quality are: database management systems, data dictionaries, report generators, graphics products, software monitors and optimizers, flow analyzers, cross-reference generators, languages, preprocessors, debugging software, program and test data libraries, an interactive programming facility, file maintenance systems, and project management systems.

Some of the techniques which can be utilized to enhance quality include structured walkthroughs, structured testing, development support libraries, excessive training, follow-on consulting, project audits, and software quality checklists.

The third and perhaps the most important consideration for ensuring the design of quality software is the (3) INTELLIGENT MANAGEMENT OF PEOPLE. The best structured systems development methodology combined with the best tools and techniques will be of little value if there is a lack of appropriate management.

Demonstrate leadership by example. Show that you care. Clearly define and communicate strategy and goals, but don't define every tactical step. Encourage your employees to participate in planning tasks. Allow them to be a part of the decision-making process. Ensure that they have adequate professional training. Provide them with feedback and recognition. And develop a constructive spirit of teamwork and cooperation.

To summarize, the three factors which most greatly influence the design of quality software are: the use of a structured systems development methodology, the use of quality-enhancing tools and techniques, and the intelligent management of people.

BIBLIOGRAPHY

Cho, Chin-Kuei. An Introduction to Software Quality Control. John Wiley & Sons, Inc., 1980.

Crowley, John D. "The Application Development Process: What's Wrong With It?" 1981 ACM Workshop/Symposium on Measurement and Evaluation of Software Quality. JDC Associates, 1979.

McCall, James A. "An Introduction to Software Quality Metrics." In Concepts of Software Quality, 1978.

Myers, Glenford J. Software Reliability Principles and Practices. John Wiley & Sons, Inc., 1976.

Myers, Glenford J. The Art of Software Testing. John Wiley & Sons, Inc., 1979.

Welburn, Tuler. Structured Cobol Fundamentals and Style. Mayfield Publishing Company, 1981.

Zachmann, William F. Keys to Enhancing System Development Productivity. Amacom, 1981.

Characteristics of Software Quality. TRW Systems and Energy, Inc., North-Holland Publishing Co., 1978.

"Program Design Techniques." In EDP Analyzer, March, 1979, Vol. 17, No. 3.

"The Production of Better Software." In EDP Analyzer, February 1979, Vol. 17, No. 2.

Software Product Lifecycle. Hewlett-Packard Company, March, 1981.