A technique for comparative assessment of software development management policies

by BRENDAN D. L. MULHALL and STEVEN M. JACOBS
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

INTRODUCTION

This paper describes a technique designed for organizing and structuring the comparison of software development management and software design practices. It is intended to provide a general method for assessing proposed software practices, especially of bidders on software contracts, and also to provide a visual aid in explaining to management how these proposed software practices comply with specifications, exceed specifications or are lacking.

An evaluator confronted with a request for proposal (RFP) or the proposal response to an RFP is often confronted with an enormous amount of documents that contain software policies, standards, and guidelines. Frequently, the software specification in the RFP or proposal response is not well organized and difficult to assess. To aid the evaluator in organizing his or her thinking and assure the completeness of the review, it is valuable to have a structured, disciplined approach to accomplish this evaluation.

This paper outlines a method of “getting started” with the evaluation process. It describes a technique for structuring the evaluation and illustrating completeness of software specification in the RFP proposal. This technique also aids reviewers of the final evaluation. Both the organization being evaluated and the evaluator’s critiques and policy standards are checked.

The primary intention of this paper is not a summary of any of the referenced documents. Rather, this technique highlights those practices and policies which are considered valuable because they increase computer programmer productivity, reduce software life cycle costs (i.e., development and maintenance costs), and increase project management visibility. This effort was performed in support of the U.S. Army Remotely Piloted Vehicle (RPV) Project for the Aviation Research and Development Command (AVRADCOM), St. Louis, Mo, through the Defense Technology Office within the Technology Development Program at the Jet Propulsion Laboratory (JPL), California Institute of Technology. Consequently, U.S. Army and JPL software policies are used as an example, even though the assessment technique is completely general.

ASSESSMENT METHODOLOGY

The technique is represented in the form of a matrix labeled Table I. The major row identifiers are nine key areas of computer software development methodology and management procedures. They are: Standards Required, Roles and Assignments, Documents, Planning, Testing, Reviews, Change Control, and Deliverables. Classical components of the software project life-cycle as defined by DeMarco [1] such as requirements analysis, design, coding, etc. are included within these rows.

There are three columns in Table I. A leading or identifying column is titled “Generic Names.” This column categorizes the rows within each major category of the table. The generic names provide a consistent breakdown of the details of each major area of the software practice so that a complete evaluation is accomplished. Two blank columns follow which will be filled in as part of the policy assessment procedure.

The assessment of any software management policy is a straightforward procedure. First, a review of the software development policy under scrutiny is performed. Secondly, the policy to be used as a standard or baseline is broken down by generic name items and entered in its column. A column representing the bidder’s policy is constructed and entries are made for each row (see Table I). Each item in the two columns is a specific detail of the two policies. The two policies are then compared, row by row. Concurrence with accepted software management policies as specified in a Request for Proposal, for example, can be determined by inspection, by noting where there are differences between the two columns. The degree of compliance is easily illustrated to any technical audience by use of the table. Shortcomings can also be highlighted. This degree of compliance can evolve into a more formal statement of adequacy of the given bidder’s policy, to be used as a final decision point for award or modification of a software contract.

In evaluating proposals, each bidder’s procedure would be compared row by row. Where there was no discussion, the bidder would be downgraded for lack of understanding. Where there is over specification, this would be noted as a
possible cost reduction during the implementation or as a result of contract negotiation. Thus, Table I is not merely a comparison of management policies of two selected software development policies. Rather, Table I can be expanded and used as a tool for analyzing any number of software development tasks, comparing given management practices with a variety of accepted standards and the software engineering literature.

APPLICATION OF THE TECHNIQUE

An example of the application of the technique is shown by the matrix contained in Table II. The first two columns of the matrix are two U.S. Army procedures for software design practices and management policy. The first is a proposed procedure [2] (Army regulation) and the second, the procedure specified in the RPV Request for Proposal, [3,4]. The next two columns are software procedures which have been used at one time or another at JPL. These are the procedures for the National Aeronautics and Space Administration (NASA) Deep Space Network (DSN) [5-11], and the Department of Energy sponsored Vehicle Economy, Emissions, and Performance (VEEP) computer simulation project [12]. Finally, the last two columns are titled "The Literature" and show four reference texts which represent a sample of the academic viewpoint written by Tausworthe [13,14] and Yourdon [15,16].

In some instances, the generic names simply do not apply across the board and, consequently, a bracket is used to communicate the collapse of these identifiers. In this case, a short description is used instead which is amplified in the text.

The nine key areas of software development and their concomitant generic names are described below as are the entries for the four policies and four texts used for the example.

Standards required

In this major area, those standards which are fundamental to modern software engineering according to Tausworthe and Jensen and Tonies [13,14] were listed. These are top-down design, structured programming, modularity, documenting...
with design, testing during coding, the preferred use of high-level languages, and standards for firmware. The applicability of each of these is indicated across the row which is intended to show that it is a recognized attribute of the particular standard practice.

### Roles and assignments

In this area, the concepts and semantics become something of a problem because every organization has its own names. For example, the DSN lists Cognizant Design En-

---

### TABLE II. Example of software development management assessment (continued)

<table>
<thead>
<tr>
<th>Generic Names</th>
<th>Proposed</th>
<th>Deep Space Network</th>
<th>VEP</th>
<th>JPL</th>
<th>THE LITERATURE</th>
</tr>
</thead>
</table>
| Requirements, | System specification | To be defined by contractor in CPDP | Software Requirements Document | In Research Objectives Plan | Software Requirements Document

### PLANNING

- Accountability to be defined by contractor in CPDP
- Computer Resource Management
- Project Control
- Specification in CPDP
- Space, memory, & logic management Plan
- Specific in DSN Software Management Plan (CPDP)
- Schedule established with shared concurrency of development activities
- Complete management procedure for each version or major modification

### TESTING

- Top-Down Testing
- Design of Test
- Test Data
- Test Tools
- Error Handling
- Error Detection
- Error Handling
- Acceptance
- Integration
- In-Field and Field Test
- Required
- Required
- Required
- Required
- Required
- Required
- Required
- Required
- Required
- Required
- Required
- Required
- Required

---
TABLE II. Example of software development management assessment (continued)

<table>
<thead>
<tr>
<th>Requirements, System Specification</th>
<th>U. S. ARMY</th>
<th>JPL</th>
<th>THE LITERATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Plan, CRDP</td>
<td>CRDP</td>
<td>CRDP</td>
<td>Once</td>
</tr>
<tr>
<td>High-level Design, Module by module</td>
<td>Module by</td>
<td>Module by</td>
<td></td>
</tr>
<tr>
<td>Detailed Design, Module by module</td>
<td>module</td>
<td>when requested,</td>
<td></td>
</tr>
<tr>
<td>Acceptance Test Procedures, CPDP</td>
<td>CPDP</td>
<td>CPDP</td>
<td>Once</td>
</tr>
<tr>
<td>Operations Manual, CPDP</td>
<td>CPDP</td>
<td>CPDP</td>
<td>Once</td>
</tr>
</tbody>
</table>

| CHANGE* CONTROL                   |                      |     |                |
|-----------------------------------|                      |     |                |
| During development,                |                      |     |                |
| To be defined by contractor in CPDP | Controlled by software implementation team and maintained by secretariat | Team agreement on a module basis | Change control cycle established when problem occurs affecting software development library elements under configuration control |
| After transfer to operations,      |                      |     |                |
| Software Configuration Control Board | By formal change control | Sponsor's responsibility with support | Omitted |

| DELIVERABLES*                     |                      |     |                |
|-----------------------------------|                      |     |                |
| Documents, All software documents | All documents        | All documents | All documents |
| Code, Software and operating system | High level language and machine code | High level language and machine code | High level language and machine code |
| Test Results, According to approved test plan | Required | Required | Required |
| Test Data, Required               | Required | Required | Required |

| SOFTWARE DEVELOPMENT COSTS        |                      |     |                |
|-----------------------------------|                      |     |                |
| None                              | N/A                  | TBD | $8 million     |
| $250,000                          | N/A                  | N/A |                |

Software Development Management areas that require particular attention

gineer (CDE), Cognizant Sustaining Engineer (CSE), and Cognizant Operations Engineer (COE) for the key software tasks of design, maintenance, and operation, respectively. Under "The Literature," Yourdon has several interesting and even amusing names for roles which are identified in other systems. This text identifies roles which exist in every computer facility though perhaps not explicitly recognized, such as "Tool Smith" and "Language Lawyer."

Discussion of common practice with DSN personnel was required to determine the equivalents to the "Tool Smith," in their policy of requiring a single development of utility software (such as particular I/O handlers or character-code conversions) that will meet identical requirements in a number of programs. On the other hand, the function performed by Yourdon's "Language Lawyer" is often not centralized, but is duplicated many times over by programmers who individually maintain sizable files of memos and other correspondence that tell how the operating systems and compilers really work.

The effects of these differences in terminology cause significant expenditure of effort to determine the real correspondence between role statements, efforts that are not permitted in the evaluation of competitive bids. This is a problem area in the use of the technique described here and requires considerable judgment by the users of the technique.

Software documents

Adequate documentation of software is imperative. The Documents row outlines the names of required documents that accompany software development. For example, the DSN requires the Software Requirements Document (SRD) which contains a statement of those system functional requirements which are to be implemented in software, Software Design Document (SDD) the top-level architectural design, Software Specification Document (SSD) which contains the detailed design and which becomes a maintenance document, Software Test and Transfer Document (STT) for describing test and transfer to operations procedures, and Software Operations Manual (SOM) the user guide. Other projects require Software Functional Description (SFD), which includes functional requirements, and Software Test Report (STR), which document test results as well.

There is a conscious effort in the selection of generic document names in the table to make them relate in a nearly
Planning

Planning is, of course, a key step in any project implementation, and imperative in software development. The approval of a well-defined software management and implementation plan is a common practice at JPL and essential for software development management. Even small software tasks require planning in order to meet schedule and functional requirements. Two items of planning, namely resource allocation (the employment of personnel and other resources that we measure in dollars such as computer time, graphic support, etc.) and computer resources, (the amount of core, CPU time, peripherals, etc. which any program shares with other programs while co-existing in the same software effort), may seem redundant but are not.

In the allocation of resources, the life cycle cost of the system is of primary importance. The life cycle costs consist of the design and implementation costs as well as the operating and maintenance costs through the useful life of the system, and, finally, archival storage costs after replacement. To forecast life cycle costs one must consider both the software and the hardware together as a complete operating system. For example, if a program must be squeezed down by partitioning into segments to fit into a limited amount of main memory, as opposed to designing with an adequate amount of memory to reasonably accommodate the program and also to allow for some growth, then both implementation and maintenance costs may be increased. Similar tradeoffs may exist such as I/O and the choice of peripherals.

Testing

Ultimately, all useful computer programs must be shown to be both correct and valid. A "correct" program has internal consistency within its architecture, logic, syntax, and nomenclature. A valid program satisfies the functional requirements of the user. Like top-down design, top-down testing is often, but not always, according to Yourdon and Constantine16, employed to gain the full benefit of the top-down approach. Procedures for testing software must be designed in advance. Definitions of test data, test tools, and the use of stubs as in Yourdon13,16 are also required. Procedures to be followed based upon the test results must also be defined. The terms verification and validation must clearly be understood by both the software bidder and customer.

Structured walk-throughs, as described by Yourdon [15,16], are a key means of ensuring software consistency and correctness. The criteria for software acceptance must also be established. The integration of a software system requires additional test procedures, occasionally examining software in an in-plant versus field use test, or over the entire life cycle of the system.

Reviews

This section uses the same generics as the documents and the roles and assignments sections to show this relationship.

Change control

Any change to computer software can have drastic effects upon delivery schedule and computer program validity. Changes that occur during or after the software design and implementation require change control procedures. Therefore, the comparisons of change control procedures in Tables I and II are divided into two areas, the first being the change control during the development period and secondly, after initial acceptance of the software by the user. These are separate and very distinct areas requiring different procedures. Changes that affect interfaces being worked on by different groups require careful control at all times after the interface definition has been established, even during implementation. Changes that are found necessary after design review of architectural or high-level designs need not be as formally controlled since the software has not been accepted by the user and disseminated through the user organization. Authorization of the change at this stage still requires the same approval cycle as the original design.

When properly applied, top-down design, modularity, and structured programming techniques all help to minimize or prevent the change from rippling through the system design by isolating each function in its own module and by eliminating sneaky paths which allow changes to cause unforeseen effects.

An automated software development tool such as SDDL (Software Design and Documentation Language), developed at JPL, can aid in tracking the changes that occur during the design process itself by making it easy 1) to record such changes and 2) to see where changes have occurred. SDDL also lends itself to software implementation using structured programming techniques, minimizing the effects of changes when structured, modular computer programs are constructed [19].

Changes that occur to software after delivery require formal change control procedures that not only update the software, but also inform others by issuing a notification of change and change pages to holders of the software documentation and, update operators' manuals.

Deliverables

Not all projects call for the same items as deliverables. For example, the requirements and the high-level design documents are not part of the standard deliverables for the DSN because they were not deemed necessary to survive the implementation. Also, not every standard practice calls
for delivery of both machine code and high-level language though this is an essential item for maintenance purposes. Operating systems are often required as part of software delivery. The provision of operations and maintenance manuals also are handled differently by different projects.

In the item with the generic name Code, there are two levels specified. High-level language or source code can be a program design language or a compiler language such as FORTRAN or COBOL. Even assembly language, if that was the highest level language used in the development, can be the source code. The most human-oriented language employed is the source code and should be included as part of the product delivered to the customer, since it is needed to maintain and revise the program. The second level is machine code representing the entire working program.

Software development costs

This last row in Table II indicates the relative magnitude of the software implementation projects. It is recognized that complete, rigorous software practices are not always appropriate in small projects. The most efficient approach is to use those aspects of software methodology which are valuable for control, design, maintenance, and survivability of the software system while not requiring all of the formal reviews, documentation standards, etc. which might be required to manage a larger project. Therefore, keeping in mind the size and scope of the programming task while utilizing the assessment technique is important.

If more than two procedures are being compared, relative development costs become an interesting item and, hence, are included in Table II but not Table I.

COMPARISON OF POLICIES

An inspection of Table II indicates the comparability of the six approaches to software management and design. The most similar are the Deep Space Network Policy and the texts by Tausworthe which is due to the contribution that Tausworthe made in developing DSN standard practices. The Vehicle, Economy, Emission, Performance program was based heavily on the texts by Yourdon and, consequently, is a specific application of these texts.

The two Army procedures do differ somewhat. The specified procedures in the RPV RFP are based on two references, one (the RFP) is a collection of statements and various Army documents which formed the specification. The proposed Army practice [2], is more of a computer hardware resource management procedure, although it has a number of requirements on software procedures. Consequently, these two columns are not entirely comparable, but were included to show both the present Army philosophy and one possible future approach.

One area of comparison between proposals and standard practices which is not delineated in the table, is to the advantage of the software bidder is the clarity, conciseness, and discernibility of statements of the required procedures and practices.

A FINAL NOTE

Due to the differences in software policies and guidelines combined with the lack of maturity of the field of software engineering, a fair and just comparison of a number of software proposals or policies can be difficult. One must gain a substantial familiarity with the proposals or policies to be evaluated to secure valid comparison data for this technique. Special treatment of information in regards to particular proposals or policies is often necessary and slight modifications of the standards displayed by this technique may be in order. A weighting factor may also be incorporated into the table by adding another column to Table I for a multiplier or weighting-type function, or simply being more critical of certain areas and not in others, in a more subjective approach.

When there is adequate interaction between the software practitioners and a group of knowledgeable evaluators, the assessment technique can be a useful tool for highlighting the differences between approaches and for providing a basis for determining how approaches could be modified or approved.

CONCLUSION

Any bidder's proposal can be evaluated by constructing a new column and filling in each area the names or the existence or the absence of certain documents, reviews, assignments of personnel, etc. By this means the conformity of the proposed standard practice to existing conventional wisdom can be related. Shortcomings and better procedures in certain areas can also be readily identified and communicated.

The assessment technique can assist in giving structure to the process of comparing bidder's proposals, management policies of different projects or organizations, or to evaluate proposals for software management approaches.

ACKNOWLEDGMENT

This paper presents the results of one phase of research conducted at the Jet Propulsion Laboratory, California Institute of Technology for the United States Army Aviation Research and Development Command, by agreement with the National Aeronautics and Space Administration.

BIBLIOGRAPHY

Software Reliability

The first session of the software reliability area will address Software Reliability Needs. It includes three invited papers that deal, respectively, with the origination of reliability requirements, with issues of reliability measurements, and with reliability modeling and prediction. All of them represent the cutting edge of the current technology and treat the subject in a broad manner that may be of interest also to the non-specialist; e.g., the discussion of reliability requirements is based on software used for a fly-by-wire system for transport aircraft. In this context, the overall aircraft safety regulations are examined, and the derivation of computer and software reliability requirements from these is outlined. All papers recognize that software reliability is an evolving discipline, one in which we are all students, and a field of tremendous importance for the future of computers in our society.

The second session addresses Current Trends in Software Reliability, and it is structured as a panel discussion. The panelists work in fields that make utmost demands on the reliability of computing systems, and they will relate the current practices that are being used to meet these requirements. Management techniques, personnel selection and training, and technical methods in requirements analysis, design practices, structured programming and innovative test strategies will be discussed.