A survey of current practice in aerospace software development

Robert T Plant and Panagiotis Tsoumpas
Department of Computer Information Systems, University of Miami, Coral Gables, Florida 33124, USA

This paper integrates the premise that current software level practices within the aerospace industry are weak and that there is a lack of rigour in both technical and managerial areas. Results from a survey of practitioners are presented that indicate a lack of information interchange exists and that the use of formal techniques is limited. The paper proposes that this is indicative of poor life-cycle practices and that more rigorous methodologies, ones that integrate formal methods with quality practices, are required. A two-level model is proposed to address the issue.

Keywords: aerospace software, software process modelling, software practice

This paper examines the current state of the art in aerospace software development, and describes a new software development model aimed at improving the industry practice, addressing those problems raised through a survey of aerospace software practitioners.

There are many different life-cycle models in use today, including the traditional waterfall-based model of Royce1, the Spiral Model2, CASE environments, prototyping3 and formal methods4. These models are utilized in conjunction with the procurers' requirements and standards such as those proposed by ANSI, IEEE, DoD, and American Institute of Aeronautics and Astronautics (AIAA).

In addition to the development paradigm followed for the creation of a software system, there is a managerial aspect to be addressed. Management requires a mechanism for controlling the development process. This can be achieved through a model, based on the development paradigm followed for the specific project, and which specifies certain documents as outputs of each phase. These documents can then be reviewed and assessed for conformance to requirements, a verification process. When it has been established that they conform with what was expected from that specific phase, they are noted as being complete. Sommerville5 criticized such systems for making the development process less flexible towards the adoption of changes since employees are reluctant to recreate 'finalized' documents, and he argues that such systems increase the project cost since they require a great number of reports to be created without helping the effectiveness of the process. The Department of Defense Standard STD-7167A6 and the 'V diagram'7 are examples of such a model-based development philosophy. Both of these methodologies describe the development process in detail and require documents to be produced for every phase and subsequent reviews to be performed; others have gone further and detailed subprocesses and the documentation to be associated with them, e.g. the Department of the Army Software Test and Evaluation Guidelines8.

These methods of software development have a common shortfall: the absence of process control. These models clearly defined what should be done in every phase and what the resultant of that phase is; however, they did not mention how each phase's performance should be controlled in order to increase quality through measurement and management. This major weakness is, we feel, a direct contributing factor to the difficulty of controlling software development and quality, which, according to the General Accounting Office, led to only 2% of a sample group of $6.77 million in software projects being used as delivered9, (see Table 1).

In order to illustrate these difficulties, we present the results of a survey of aerospace software developers' practices in the next section. This is followed by a new software development paradigm that aims to address these issues through process control and the theories of quality management.

Industry practices and state of art report

Research objectives
The following are the objectives of the study:
(1) Identify levels of management awareness with regard to changes in technology
Aerospace software: R T Plant and P Tsoumpas

Table 1. Software delivery and usage

<table>
<thead>
<tr>
<th>Software used after extensive rework</th>
<th>19%</th>
<th>$1.3m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software delivered but never used</td>
<td>47%</td>
<td>$3.2m</td>
</tr>
<tr>
<td>Software paid for but never delivered</td>
<td>29%</td>
<td>$1.95m</td>
</tr>
<tr>
<td>Software usable after changes</td>
<td>3%</td>
<td>$0.198m</td>
</tr>
<tr>
<td>Software used as delivered</td>
<td>2%</td>
<td>$0.119m</td>
</tr>
</tbody>
</table>

(2) Identify impact of technology change upon product design.
(3) Determine the impact of the customer upon product design.
(4) Identify how management views corporate planning upon the products quality.
(5) Determine the use, standards and practice of:
   - software metrics;
   - cost models;
   - formal methods;
   - quality.
(6) Identify the level of management/technologist communication present within the organizations.

Research methodology

The methodology includes:

(1) Developing a questionnaire to meet the study objectives. There were four major sections: firstly, an examination of managerial issues, including the impact of technology changes upon product development. Secondly, the impact of formal techniques in software development. Thirdly, the impact of quality techniques. And fourthly, issues relating to the information systems department.
(2) Pilot-testing the questionnaire to a set of eight software development professionals, including a vice-president for system development for a major University, as well as a number of computer professionals dealing with software development. The questionnaire was adjusted according to the test group's comments.
(3) Mailing the questionnaires to a group of aerospace software developers. The sample population was identified as those attending the American Institute of Aeronautics and Astronautics conference Computers in Aerospace IX, San Diego, California in October 1993. The sample population was determined to be highly qualified as the respondents were all presenters of academic, jouried papers on computing issues relevant to the aerospace industry. The population did not include any members whose primary affiliation was as a faculty member at a University; also no students were included in the survey.
(4) The responses were analysed using a statistical package.

Survey results

We mailed out 152 questionnaires and received back 36 completed questionnaires for a rate of 24%. Questions 1–3 were to identify the scale of projects that the organizations were involved with, which are summarized in Table 2.

In relation to the figures in Table 2, the average number of projects that the respondent companies had are illustrated in Table 3.

Managerial and technical issues

The first section of the questionnaire examined how closely respondents followed technical developments in the fields of hardware and software. This was examined through questions 4–9.

The respondents felt that changes in hardware technology (91%, 30/33, c4.7*, (see Figure 5)), and software technology (94%, 31/33, c4.7), (see Figure 6), have a very important impact on the development of the software within their companies. They, however, followed changes in software technology (82%, 27/33, c5.7), (see Figure 3), more closely than hardware (76%), (see Figure 1). This may have been due to bias in the target group who were presenting papers at Computers in Aerospace IX, a chiefly software orientated conference. These results also reflect the nature of the industry, in that the aerospace industry itself has a significant impact upon the software industry, creating its own languages such as NASA's CLIPS*, and having technical input into language standards such as ADA 9X. The industry also creates its own specialized hardware and software to its own specific needs. This may be one

Table 2. Project scale of respondents

<table>
<thead>
<tr>
<th>Time needed to complete project (days)</th>
<th>Small project</th>
<th>Large project</th>
<th>Average project</th>
</tr>
</thead>
<tbody>
<tr>
<td>People involved</td>
<td>2</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>Resources required ($000's)</td>
<td>54.14</td>
<td>4852</td>
<td>620</td>
</tr>
</tbody>
</table>

Table 3. Companies average project load

<table>
<thead>
<tr>
<th>5 year average</th>
<th>163</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 month average</td>
<td>40</td>
</tr>
</tbody>
</table>

Figure 1 How closely do you follow changes in hardware technology?

*The percentages are cumulative over a number of categories of the query range; these categories are noted in the following way: x%, respondents/total respondents for the specific question, and cN.M where N and M signify the first and last category included in the group.
Figure 2  How often do you go to conferences with a computer hardware focus?

Figure 3  How closely do you follow changes in software technology?

Figure 4  How often do you go to conferences with a computer software focus?

Customer support issues

The second section of the questionnaire examined the relationship between the customer and the organization and how management perceived the impact of customer support upon system development. This was examined through questions 10, 11 and 12.

The second section shows an almost even split in the respondents between those companies that do and do not provide customer software support services (see Figure 7). Of those companies which do provide support, 70% (12/17, c5..7), (see Figure 8), of the respondents felt that this service was important to its success. These respondents went further and 75% (12/15, c5..7), (see Figure 9), stated that they utilize the customer input in the development of future products, and 62% (10/15, c5..7), (see Figure 10), utilized the input to influence the product design process. The inter-relationship between customer support and software development for those respondents who answered negatively to providing software support was not explored by the survey. However, it may be that the software is developed for in-house applications in many instances and for highly sophisticated clients who represent corporations...
Aerospace software: R T Plant and P Tsoumpas

Figure 7 Do you provide a customer support service (help desk) for the software that you develop?

Figure 9 To what extent is the customer support service important for the success of your software products?

Figure 8 To what extent is the customer support service utilized in decisions about future software products?

Figure 10 To what extent is the customer support service utilized in decisions about process design?

Informal reporting channels
The third section considers the role that informal channels of reporting have in relation to the software design and development process, and how that reporting is conveyed to management within the organization.

In order to obtain an overview of reporting channel usage the survey asked, in question 13, which reporting channels were utilized (see Figure 11). This indicated the importance of informal reporting channels, i.e. through people whose opinion is important even though they may not be an established part of the organization's decision-making hierarchy. A closer examination of the usage of informal inputs clearly indicated that respondents felt they are important both in making decisions about future software products (78%, 25/32, c5..7), (see Figure 12), and process design (70%, 23/33, c5..7), (see Figure 13).

Software metrics
In this section we will consider the usage of software metrics amongst the respondents. These correspond to questions 15 to 22.

An overview of the usage of general metrics was established through questions 15, 17 and 18, in which it was asked whether the respondents had any software metrics and if so, what they were. Figure 14 indicates that 80% (26/33, c5..7) of the respondents had some type of software metrics, with the most popular being the number of bugs found per month (76%, 25/33, c5..7), followed by the number of defects found per month (70%, 23/33, c5..7), and the overall defect rate (69%, 22/33, c5..7).

Figure 11 What are the types of communications channels that your company utilizes for new product requirements and/or requests?
Figure 12 How important is informal input for decisions about future software products?

Figure 13 How important is informal input for decisions about process design?

was established that 64% (21/33), (see Figure 17), of respondents have a software metrics program in place and that a significant number of respondents indicated that they collect metrics on program size (54%, 18/33), program productivity (45%, 15/33), errors/LOC (30%, 10/33) and

Figure 14 Does your company collect information about programmer productivity, errors per line of code, etc.?

Figure 15 To what extent do you utilize information about performance metrics in decisions about projects (staffing and time cost estimations)?

Figure 16 To what extent do you utilize information about performance metrics in decisions about human resources (promotions, career advancement opportunities, etc.)?

time to repair errors (27%, 9/33), (see Figure 14). In order to determine the usage of these metrics, two further questions were asked in which 76% (19/25, c4..7), (see Figure 15), of respondents indicated that these metrics were influential in making decisions about project staffing, time and cost estimates. However, 52% (13/25, c1..2), (see Figure 16), of respondents felt that the metrics were not at all important in determining human resource decisions.

In question 18 (see Figure 18), the respondents provided detailed indication of the techniques used, in which program size (62%, 13/21), McCabe’s Complexity measures (43%, 9/21) and function points (28%, 6/21) were most widely used. However, even these were utilized by only 64% (21/33) of all respondents. The reasons provided by the respondents’ low usage of metrics were lack of training where 58% (18/31, c5..7), (see Figure 19), felt this is of above average importance, and lack of tools, where 74% (22/31, c5..7), (see Figure 20), felt this significant.

Software cost modelling

In this section we will consider the usage of software cost

Information and Software Technology 1995 Volume 37 Number 11 627
models amongst the respondents. These correspond to questions 23 to 27.

In question 23 (Figure 21), 60% (20/33) of respondents indicated that some form of cost and time estimate process is in place within their organization, the primary process (90%, 18/20) utilized being 'expert judgement', with only a low number of respondents (35%, 7/20) using the COCOMO model and none SLIM (Figure 22). Further, 63% (12/19, c1.3), (see Figure 23), of the respondents indicated that their companies were poor at following up on reasons for the discrepancies between actual results and the original estimated values. On occasions when follow-up did occur, question 27 investigated what are the two most common actions with respect to the discrepancies; the following feedback was amongst the answers obtained:

Q27: What are two most common actions with respect to cost/time estimation discrepancies?
1. Go back and renegotiate with the customer
2. Revise future estimates
3. Evaluate development process
4. Attempt to understand failures in design process
5. Attempt to improve customer input process
6. ‘Lessons learned log’, analysis of lessons learned
7. Management review of problems
8. Model tuning
9. Cause factor analysis
10. Blame the workers

Figure 17 Do you have a software metrics program in place?

Figure 18 What are the software metrics that you use?

Figure 19 To what extent do you believe that lack of training obstructs the implementation of a software metrics program?

Figure 20 To what extent do you believe that lack of automation, i.e. tools for collecting the required information, obstructs the implementation of a software metrics program?

Figure 21 Do you have a formal or informal cost and time estimation process in place?
What is the method that you use for cost estimation?

Does your company follow up on reasons for discrepancies between actual results and cost and time estimation?

11. Cite change of scope
12. Modify guidelines
13. Ignore if special circumstances
14. Adjustment of estimation analysis rates and weights
15. No action, just informational
16. Lessons learned reports go to process team that decides on changes if any to estimating tools
17. Change development process
18. Modify model estimations

A wide range of answers are presented. Some of the answers are particularly interesting since they show how by following such suggestions, e.g. 2, 14, 18, based only on the last experience, without any consideration to possible reasons for the deviation, the estimation process could evolve into the process of chasing a moving target. Comment 10 reinforces the notion that presented in the wrong way and in a corporate culture adverse to change, metrics can easily be perceived as a significant threat by the worker.

We can also see some good suggestions where these discrepancies are considered part of the system process and are studied in order to determine the cause.

**Formal methods**

In this section we will consider the role of formal methods.

In answering question 28, (Figure 24), the respondents indicated overwhelmingly (93%, 28/30) that they did not utilize formal methods of software development and 73%(22/30), (see Figure 25), stated that they did not utilize a technique for guaranteeing conformance to customer specifications.

Quality issues

This section considers the usage of quality assurance techniques within an organization both at managerial and technical levels. This is done through questions 30 to 35.

In the area of quality assurance, 60%(18/30), (Figure 26), of the respondents indicated that their organizations had a quality program in place, and that 70%(14/20), (Figure 27), of the software development personnel are actively involved within that program. Of those respondents involved in quality there was an even split amongst those who do and those who do not utilize quality circles. The respondents overwhelmingly stated (93%, 15/16), (Figure 29), that they did not utilize the PDCA cycle; however, we feel that this may be due to the survey using the acronym 'PDCA' rather than state 'Plan Do Check Act' as this cycle is a central tenet of most quality philosophies.
Even though 60% of the participants indicated they utilized a quality program, 71% (15/21), (Figure 30), of the respondents indicated that their company does not utilize employee rotation within the organization; this may be a reflection upon the corporate philosophy, and maturity of the company in terms of the total quality management culture. However, 71% (22/31; c4..7), (see Figure 31), of respondents felt that training was an important aspect in obtaining advancement within their organizations. The training function may be taking the place of the training that would be obtained in the employee rotation aspect of the quality program.
Overview: The MM-level process model

In the second section we examined several aspects of aerospace software development, considering both technical and management issues. This investigation indicated that there exist areas of weakness within the development process and that there needs to be more integration and formality. One reason for this situation may be the lifecycle methodologies utilized, as the traditional stage-based waterfall type models do not take into account development in areas such as cost modelling or formal methods, and the rigorous approaches such as those advocated by the formalists are often perceived to be incapable of integration with other aspects of system development and difficult for management to control. In order to overcome these issues we propose a model for software development that utilizes and integrates techniques both at the managerial and technical levels. The model attempts to fulfill the needs of the aerospace industry as recognized by the survey, whilst being general enough to be applied to other domains.

In the remainder of this section we will describe a two-level software development life-cycle methodology. These levels: Macro and Micro, combine to compose what we term the MM-level model. The Macro perspective places the software development process in its environment with respect to the external factors. The second level describes the software development process from a Micro perspective, where the methods and practices of improving the quality of products and processes are of primary concern.

Macro perspective

The proposed Macro perspective views the software development process as a part of a greater system that includes factors external to the development process itself. These external factors are:

- Latest developments in the hardware and software areas.
- The traditional input from customer about requirements of the system under study.
- Input from various customers, through customer support, about problems discovered during the operation stage, and from potential customers, through marketing research, about current needs.
- Input from top management, through the CIO, about corporate business plans.

The influence of these external factors on the software development process takes place through various forms: directly in the form of user requirements, or indirectly by changes and developments in technology. Even though the indirect factors have great impact on the development process, their importance has historically been neglected or downplayed. The model we propose utilizes these external factors in order to increase the awareness of management about such factors. We will now consider these external factors in more detail and their effect upon the development process.

The DeltaT (ΔT) effect

The rapidity with which technology changes in the area of information technology means that management and software engineers cannot afford to isolate themselves from these changes. However, unless management and developers control their software processes through an understanding of the impact that hardware, software, practical and theoretical developments have upon it, then these technology changes may have a serious and detrimental effect upon the developers' software creation process. We call this the ΔT effect (change in Technology), and suggest that it is management's role to positively encourage the dissemination of new knowledge and techniques.

The ΔT factor was investigated in our survey through several questions: number 46 asked to what extent there had been an infusion of information technology within the
respondents’ company, in terms of importance, impact and significance, to which 88% of respondents indicated that the impact was significant [C4..C7]. (see Figure 33). This result was reinforced by the answers to questions 4 to 9 where the respondents indicated that changes in hardware and software significantly impacted the software developed by the company. Thus, we can see that AT has a significant impact upon not only the technical development of a product but also upon the management of the organization who needs to be aware of, and adaptive to, the changes in technology in order to ensure their competitiveness.

Customer support Customer support is a function of a software development organization, whose input is either not fully utilized or it is neglected altogether. Customer support serves a dual purpose: (a) it supports customers during the implementation with training; and (b) it is the frontline of the corporation for obtaining input from customers about discovered defects.

The survey indicated (question 10) that 55% of the respondents do not provide customer support. As stated earlier, this may be due to the in-house nature of the software developed, the sophistication of users or a very close working relationship between the developer and user, all of which are closely related to the nature of the industry and the level of technology used. Those respondents that indicated they provide customer support services, however, then indicated the importance of this service both directly (Q11) in that 13/17 [C4..C7] view customer support important for the success of their software product, and indirectly in the feedback achieved through customer support utilized in decisions about future software products and as input data into understanding the process design process itself. These indirect usages were utilized by 12/16% and 11/16% of the respondents respectively.

Corporate plans The corporate business plan is an external factor which has a special relationship with the end-product’s quality. If there is a corporate plan for releasing a new product by a certain date and there are delays in the progress of the project, management’s attitude will be decisive. Would management’s attitude be ‘meet the deadline regardless of defects’, or alternatively ‘continue the high quality work and let’s figure a way to work more efficiently so that we will not have any more delays’, be instrumental in determining final product’s quality? Research by Weinberg and Schulman13, has found that given specific objectives, programmers can make the required choices to meet these objectives, provided the objectives do not conflict with each other. It has been suggested by all the quality advocates12-15, that management’s commitment is of paramount importance for the success of a quality improvement program. Thus, we feel it appropriate to include corporate plans as an influential factor in the software development process. Management commitment to the process of quality is therefore imperative: we can see that they have the ability to determine the pressure placed upon the software engineers to balance criteria such as deadline dates against quality levels. Even though some research16 has discussed the influence of corporate top management in information systems planning, there is little formal research on how the top management plans influence the quality of the end product.

In order for management to be able to balance the requirements of their customers with the status of the software development, there is a strong need for open channels of communication between the managerial and technical areas of the organization. In order to determine the usage of communication channels, the respondents in question 13 stated that they utilized both formalized and informal channels but that there is a need for both a management awareness of the technical development issues and an awareness of the corporate plans by the technical staff with multi-level reporting facilities.

The awareness levels of the corporate and technical management with respect to each other’s area were examined in the survey. Figure 35 shows how 80% of the respondents felt that the IS managers were aware [C4..C7] of the companies’ business plans, and Figure 34 shows that 79% felt that the companies’ top management were informed about information technology. These are levels to be expected of a highly advanced technological dependent industry such as aerospace where management tend to have
Figure 35 How informed are your company’s information systems managers about your company’s business plans?

technical backgrounds. However, further research is necessary in order to establish the exact nature of the management backgrounds as well as how these interrelationships would emerge in other industries.

Summary of Macro perspective It has been the aim of this section to show the importance of the external factors in relation to the process of software development. This is important for two major reasons. First, the identification of influential factors provides the manager with a better understanding of these factors and their role in the software development process. Second, managers by knowing about these factors can incorporate them in their plans and control their influence, over the software development process.

Micro perspective
Having briefly considered the Macro perspective and identified the need for management and technical integration and information flow, we will now consider the Micro level of the model. The aim of this level is to provide a structure that both supports technical excellence in a quality life-cycle environment, and provides support to management regarding the software process itself. This section of the paper examines the functional areas of the Micro level: we term these the ‘life support tools’, as they support the software development life-cycle as well as the management process. We also consider the interaction of these functional areas and their place within the quality principles utilized in the model. The Micro level is illustrated in Figure 36.

Boehm* recognizes, indirectly, that the major problem of the software development process is the lack of adequate planning, therefore he suggests that risk analysis is helpful in identifying problems which might occur. We strongly agree with this argument, however, risk analysis is only one of several techniques that management should utilize in its effort to monitor the software development process, and several of these were examined in our survey. The results of the survey as outlined earlier give a strong indication that the use of a set of formalized and integrated techniques for software development would enhance the management’s planning ability.

Some of the life support tools we advocate are:

- A historical database that will hold information about various aspects of the development process, such as models of previously developed systems, models of designs of those systems and their respective specifications, measurements of product and process attributes, etc. This database should be a repository where employees should refer for information about company’s ‘experience’.
- A software metrics program that will be the guidance for measuring attributes of both product and process as well...
as for the effort of identifying new attributes and improve the program itself. Grady and Caswell\textsuperscript{17} offer a list of 10 steps which a software metrics program should follow in order to be successful.

- **Configuration management**, an aspect that recognizes that changes during a development process will occur and management is ready to treat them formally and accommodate them. Bryan and Siegel\textsuperscript{11} argue that configuration management improves the visibility and the traceability of events that occur during a project.

- **Cost estimation model.** We believe that the best way for a company to start its quality improvement program is the adoption of a cost estimation model. Notwithstanding the shortcomings of the available models, such as COCOMO\textsuperscript{9}, these models could serve as a platform for identifying important factors in the development process, until the quality program achieves momentum and the company efforts will be able to produce its own list of important parameters.

- **Quality management practices**, such as training and quality control circles (QC circles). Training is very important especially in areas that change rapidly, such as software engineering. Courses in measurement theory and statistics, as well as in risk analysis and software engineering practices utilized by the company, i.e. formal methods, will have a great impact on the employees.

In the following sections we will consider several of the tools in this life support system, identify their strengths and how they contribute in improving the quality of the endproduct.

**Requirements formation** As has been noted by researchers and practitioners\textsuperscript{19}, the initial phase in the software development life-cycle, the requirements phase, is of great importance, since by describing what a software system is expected to do, this determines the success of that system in fulfilling its proposed function.

Traditional software industry practice has been to use natural language as a medium to express the functionality of the system. However, there are many problems with the use of natural language that cannot be overcome\textsuperscript{6,21}. The weakness of natural language specifications has led researchers such as Hoare\textsuperscript{20} to investigate the use of formal mathematical systems in specification. An aspect of this research by the formalists is in the area of program correctness. This is done by describing, through mathematics, the function of a piece of program code, implementing the function and then using an axiomatic approach to prove that the program matches the algorithm specified. Ideally the code would match the algorithm and meet its correctness obligation without the necessity of resorting to exhaustive traditional testing techniques.

The ability to construct formal, unambiguous requirements definitions and subsequently correct programs from these specifications is an area central to safety critical software systems. Thus, the premise is that the aerospace industry would embrace these ideals. This was examined in question 28 of the survey in which 94% responded that they did not utilize formal methods. This, coupled with the results from question 29 where 73% stated they did not utilize a technique for guaranteeing conformance to customer specifications, is surprising. The exact reasons behind this need further research to identify why there is a lack of conformance to more exacting levels. Initial indications are that the vendors produce the software only to those standards required by the procurer, e.g. MIL-STD-2167A, and it is these standards which need strengthening.

**Measurement and metrics** In order to assess the impact of differing methodologies upon development, software engineers can utilize measurement theory and software metrics, such that they can more easily determine the impact and effect one parameter of a design has upon another. Research is active in this area, and workers consider such design issues as module length, optimal number of modules, and criteria of module separation\textsuperscript{22,23}.

The aim of software metrics is to assist the developer to understand the relationship between the parameters of software design and its creation. These parameters are, however, not always easy to determine or measure, in addition to the difficulty of determining the consequences of the results. For example, a software engineer may use the whole-function criterion or an optimum-length criterion\textsuperscript{22} to decide when to split a program in modules, or determine the optimum length of a module; but other human-based parameters such as measurement of a programmer’s experience, or suitability of the programming language used, are not so easily measured.

This state of flux in software measurement necessitates that companies utilize a software metrics measurement program that will help each specific company to understand their own development parameters and their inter-relations. Grady and Caswell\textsuperscript{17} have pointed out that a program like this is difficult to implement, and serious commitment of all interested parties should be obtained before the implementation. Successful implementation lies on the following two aspects:

1. Training courses that will help programmers, analysts, and project managers alike to understand the importance of such tools.
2. The reassurance of the top management that this tool will be used only for the improvement of the product and the process and not as an evaluation instrument.

In addition, Grady and Caswell\textsuperscript{17} point out the importance of automated data collection. The lower the degree of human intervention in the collection process, the higher the degree of accuracy in those measurements. However, some data will still necessitate manual collection, and this can be done through standardized forms, after a training course has been undertaken to reinforce their validity. Results from the metrics and data can then be used for improving subsequent projects and in establishing standards for every task. These standards can then be regularly evaluated and altered accordingly.

The literature describes a great number of software metrics gauging various attributes of the software development product. A comprehensive review of such metrics as well as predictive models is presented by Conte \textit{et al.}\textsuperscript{22} and Zuse\textsuperscript{23}. They present various metrics in categories such as
size, data structure, logic structure, effort and cost, defect, and design metrics.

In the survey, we examined the usage of metrics within the aerospace industry and we saw that 64% of the respondents had a metrics program in place. Detailed examination of the metrics used indicated that they had utilized only the most informal metrics, such as lines of code and McCabe's complexity metric. Their use of the metrics was in a mixed manner, positive in their usage with respect to managerial decisions on cost estimates and staffing times, yet negative when used in human resource decisions. This is especially disturbing when the level of metric sophistication is considered. However, the fact that the respondents used metrics and deployed them to a limited extent in a managerial capacity is an indication that with the adoption of more sophisticated metrics, training and tools, the use of metrics in the future, especially when integrated with other models, such as cost and reliability and channelled through to management levels, will have a significant positive impact upon the decision-making process. This is the approach we advocate in the MM-level model.

**Cost estimation model adoption** The first step in the development of a system within an organization is a managerial one, in that whilst the requirements for a system are being formulated the system should be endorsed and backed by the top management of the organization. This necessitates that the system perform a useful function, be revenue producing and be developed in a cost-effective manner. Thus, it is necessary for the development team to adopt a software cost estimation model such as COCOMO, SOFTCOST or COSTMODL prior to system development. This is necessary for many reasons, in that it will identify cost centres that may occur, special system development needs, such as training or equipment, and that these needs will be met prior to development, ensuring management commitment to the project from the conception of the system.

The utilization of cost models was briefly examined earlier, under 'Software cost modelling', where it was stated that 60% of respondents had a formal or informal cost-and-time estimation process in place. However, only 36% used a formalized technique, primarily COCOMO. This figure is heavily dependent upon the procurement process, and significant influencing factors are government agencies such as NASA who require cost models to be utilized for all NASA software project developments, primarily through COSTMODL, the trend being towards formalized models of cost and which in turn will require formalized development methodologies.

**Quality management practices** The quality management principal 'each process is the customer of the previous process and the supplier of the subsequent one' can be considered as tenet for the software company's process management. This can be seen when the specification of the system is passed from the requirements-formalization team to the software design team and on through the life-cycle. The specification has to possess the ability to improve the communication between all members of the development process. Thus the key to achieving this process pipeline is communication.

One approach to achieving this level of communication and interaction is through Quality Circles, where professionals from both sides can come together and discuss methods of improving the current practice by introducing new documents, changes in currently used documents, and additions in the utilized tools.

The concept of quality through circles is a central aspect of the MM-level model where each phase of development interacts with the life-support tool providing a historical database for the other aspects of the development and for future projects. This centralised focus upon quality allows other aspects of TQM to be integrated into the life-cycle i.e. employee rotation.

The use of quality in aerospace software development was considered briefly above, under 'Quality issues' where 60% of the respondents did have a quality program in place and that 70% of software personnel were involved in quality assurance, and 50% utilized quality circles. Surprisingly, 29% of respondents actively used employee rotation, indicating a segment of the survey population adhere to advanced TQM practices. The use of quality principles, being a conscious decision that the organization makes, is a significant indication of the adoption of this management philosophy by the aerospace community, a direction that is likely to continue.

**Summary of micro-perspective** The micro-perspective aims to promote the philosophy of continuous quality improvement through the utilization of life support tools and the utilization of TQM, metrics, modelling and formal methods. The model does not explicitly include a formal quality assurance phase, as this philosophy is integrated into all aspects of development. The model has also been designed to promote clarity of communication channels.

**Summary and conclusions** The aim of this paper was two-fold: firstly, to investigate the state of the art in the practice of software development within the aerospace industry, and secondly, to utilize the findings from the survey on the state of the art to assist in the creation of a two level life-cycle model for software development.

The survey predictably indicated an industry that is highly aware of and closely tied to, technology change; the industry itself being a leading proponent of technological advancement. The respondents also indicated a close association between suppliers and customers, especially in terms of customer support and technical assistance, viewing this as a very important factor in the success of a software product. Management also considered it important to utilize customer input in future product design decisions as well as influencing the process design process. This was facilitated by open communication channels, both at the formal and informal levels.

The technical aspects of the survey, however, indicated an industry that was not utilizing theoretically advanced techniques, the majority of respondents indicating that they
used only basic software metrics, such as lines of code or McCabe's measures, informal cost models, such as expert judgement, and little or no formal methods of construction. The most worrying of these were the 73% of respondents who stated that they did not utilize a technique for guaranteeing conformance to customer specification. The reason behind this needs further research, although one factor may be the close working relationship between customer and supplier, who together utilize an exploratory style to create the product. Thus the specification and traditional lifecycle model are not appropriate and hence an alternative validation process is adopted.

A more promising aspect of the survey was an indication that there was a movement towards the use of quality software practices, indicating adoption of some very advanced management practices. This is, we feel, an indication of individual corporate philosophy.

As a whole the survey indicated that the respondents were interested more in the deliverable than the process of creating that deliverable, with little thought behind the future consequences that process would have for the product, its users or future maintenance, creating a scenario that points towards those figures provided in Table 1.

The second aim of the paper was to take the findings from the survey and utilize them in the construction of a lifecycle model to assist in the creation of more rigorous and maintainable software, both at the management and technical levels. This resulted in the MM-level model. Central to the model is the ability of management to control the technological process. This can only be achieved through increased communication and understanding of the interrelationships between the processes. In order to better control the system, formalized techniques need to be adopted in development, costing, planning, etc. and this is achieved through the life support aspects of the model, which also provides the historical basis for future management decisions. This reflects a philosophy of never-ending quality improvement and control.

The survey indicated that the basic theoretical and philosophical building blocks for quality improvement are in place within aerospace software development. However, the move to total and universal commitment to these principles has not yet been made. We feel that through the enforced promotion of these standards by the procurement bodies, and the utilization of new models such as the MM-level model, significant progress towards high quality aerospace software development can be made.

References

1 Royce, W W 'Managing the development of large software systems' Proc. WESTCON, Ca., USA (1970)
3 Budde, R, Kautz, D, Kuhlenkamp, K and Zuellinghoven, H 'Prototyping—an approach to evolutionary system development Sprnger-Verlag (1992)
5 Sommerville, I 'Software engineering' 4th edn, Addison-Wesley (1992)
8 DoA PAM 73-1 Software test and evaluation guidelines Department of the Army, Pentheful 73-1, Volume 6, Washington, DC [DRAFT] (1987)
13 Jarur, J M 'Managerial breakthrough McGraw-Hill (1964)
16 King, W R 'Strategic planning for management information systems', Management Information Systems Quarterly (March 1978)
19 Boehm, B 'Software engineering economics' Prentice-Hall (1980)
20 Hoare, C A R 'Programs are predicates' in Hoare, C A R and Shepherdson, J C (eds) Mathematical logic and programming languages Prentice-Hall (1985)
21 Ince, D 'Set Piece' DataLink (23 January 1989)
22 Conte, S D, Dunmore, H E and Shen, V Y 'Software engineering metrics and models Benjamin/Cummings (1986)
24 Zuse, H 'Software complexity: measures and methods de Gruyter (1992)