Abstract

We focus on corrective maintenance carried out in the outsourced mode under strict service level agreements and present the characteristics of the problem and the activities performed. We detail the information requirements for various maintenance services, such as, emergency maintenance, production support, and corrective maintenance. We present the concept of a system execution model with its constituent nodes and arcs and present the steps to build the same. We present a case study of a large commercial outsourcing project to demonstrate how the execution model can help in making quick decisions that will reduce the turn around times of corrective maintenance requests.

Keywords: corrective maintenance, fault analysis, model, execution flow

1. Introduction

Maintenance accounts for about 40% to 90% of the software life cycle costs and perfective maintenance that deals with functional enhancements to the system is supposed to take 55% of the overall maintenance efforts [9]. In the past few years, several Fortune 500 organizations worldwide have been exploring the possibilities of outsourcing maintenance activities to software companies with offshore capabilities. Almost half of the Fortune 500 companies used Indian companies for their software needs in the year 2003 as compared to 10 in 1990. There are compulsive economic reasons for outsourcing the maintenance work [6]. The market maturity and the rate pressure on the outsourcing vendors is forcing them to look at several ways of customer retention that include cost effective delivery, strict schedule and service level adherence and meeting or exceeding the quality expectations.

The focus of this paper is on corrective maintenance carried out in the out-sourced mode under strict Service Level Agreements (SLAs). In Section 2, we review the related literature. Section 3 describes some characteristics of outsourced engagements and also the nature of corrective maintenance. Section 4 first introduces information needs in corrective maintenance and then proposes the Execution Model for tracking execution of programs and flow of business activities. Section 5 presents a case study of a large application demonstrating how the Execution Model is useful in corrective maintenance under strict SLAs. Finally, Section 6 concludes and points to further work.

2. Related Work

Details of software maintenance methodologies and processes in the industrial context are given by Pigoski [9]. There is considerable work describing models and methodologies in software maintenance [3], [8]. The IEEE standard for software maintenance [3], while discussing the phases and associated activities of maintenance, identifies project/system documentation, repository information, source code and database as the inputs for the maintenance. However, criticality of the inputs, nature of their usage and the impact of outsourcing on inputs are not highlighted. Deridder [2] proposed the usage of ontology for capturing implicit knowledge of the application domains. J. Singer [10] in his empirical research on practices of software maintenance concludes that the practitioners use source code as the primary source of information and do not necessarily trust the other documentation. Dias et al [1] in their exhaustive study propose an ontology for the maintenance knowledge by organizing it into five different aspects. Kajko-Mattson [8] has proposed a maturity model for corrective maintenance and has identified basic concepts as well as the criticality...
(degree of impact) and priority for a defect/fault. Jorgensen [4] defined an operational approach to the corrective software maintenance in which a database is created using data, execution paths and interfaces. When mission critical applications systems fail, corrective action needs to be done in minutes. We address this practical problem in this paper by proposing the concept of system execution model that will help a maintenance programmer a) to localize the problem in case of system failure and b) to take informed decisions that would minimize system down times.

3. Corrective Maintenance Outsourcing

The fixed period corrective maintenance includes multiple services as part of the portfolio such as production support, corrective maintenance, emergency maintenance and minor enhancements. It differs from the project-specific maintenance in the following ways.

• the vendor is responsible for specified types of maintenance activities for the duration of the outsourcing contract.
• The duration of the contract is usually 2-3 years which means that both vendor and the customer have to necessarily make a long term commitment towards its success.
• Applications may be geographically distributed, resulting in 24x7 support.
• The service gets measured almost every day by service levels.

Corrective maintenance deals with fixing a program or eliminating the data that is causing the problem in a very short stipulated time. This requires in-depth knowledge of the applications and their technology platforms.

3.1. Nature of the Corrective Maintenance

Production Systems under maintenance are generally well tested for simple programming errors. In the event of failure, it is difficult to trace back the problem to a specific program without reference to its context of execution. For this, the maintenance team needs to know what has happened or was happening at the time of failure. The end-user is unlikely to have this information and hence can never document it as part of the problem ticket description. Another important decision that needs to be taken is whether subsequent system execution steps can continue (whether the failure needs to be fixed or can be ignored). As each minute in the production cycle is important, quick and correct decisions made in this context would reduce the probability of SLA breaches.

3.2 Nature of Activities to be Performed

From the time a problem ticket is reported to its successful closure, the maintainer needs to perform the following activities:

• Acknowledge the problem
• Understand the problem and seek clarifications.
• Analyze the problem to localize it
• Define the solution to the problem
• Assess the impact of the solution on the other system components
• Initiate procedures for escalation, if needed
• Fix the problem and conduct appropriate testing
• Raise a maintenance request to ensure a permanent solution if the problem has been fixed temporarily.

3.3 Size of the Systems Maintained and Volume of Tickets

When a large application portfolio is outsourced and the customer business is dependent on the applications, there is often no margin for error. The number and nature of errors, at times, is large and meeting the SLAs can be very demanding. The details of a typical outsourced maintenance application system are presented in Table 1. The application running on multiple platforms is maintained by a geographically distributed team of close to 200 people. The system consists of 134 applications with a good mixture of online programs and batch jobs, as demonstrated by the number of reports and screens. Of these, 34 applications are considered to be mission critical. Analysis of the problem ticket data for three consecutive months shows that of the total number of 2777 tickets, application errors contribute almost 53%. Of the total of 1497 application related tickets, the preliminary analysis showed that about 50% of the problems are due to abnormal termination of batch jobs.

<table>
<thead>
<tr>
<th>Technology</th>
<th>No. of Applications</th>
<th>No. of Programs</th>
<th>No. of Screens</th>
<th>No. of Reports</th>
<th>No. of Jobs</th>
<th>No. of Files/Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF</td>
<td>25</td>
<td>10768</td>
<td>4069</td>
<td>3397</td>
<td>10777</td>
<td>3728</td>
</tr>
<tr>
<td>MF &amp; CS</td>
<td>33</td>
<td>6438</td>
<td>2633</td>
<td>323</td>
<td>802</td>
<td>3632</td>
</tr>
<tr>
<td>CS</td>
<td>76</td>
<td>3421</td>
<td>1409</td>
<td>359</td>
<td>165</td>
<td>1747</td>
</tr>
<tr>
<td>Total</td>
<td>134</td>
<td>21627</td>
<td>51032</td>
<td>4379</td>
<td>11944</td>
<td>25237</td>
</tr>
</tbody>
</table>

Table 1. Application Details of A Typical Outsourcing Project

MF - Mainframe; CS - Client Server
4. Modeling for Fault Analysis and Corrective Action

We held a series of discussions with project personnel who are responsible for the service delivery, work assignment and quality assurance to find the information that will help them solve corrective and emergency maintenance problems faster and better. The most important needs identified were a) information that will help localize the problem to a specific program or function of the system, and b) any information to make decisions on continuation of further execution.

4.1 Information Needs for Corrective Maintenance

Quick debugging of a problem needs precise definition of location of the problem and detailed description of its symptom. The location could be the code, database, user manual, or report. The symptom could be described in terms of the cause of the problem such as logic, computation, data, interface or even documentation. Quick isolation of the problem area requires knowledge of the following:

- Dynamic behavior of the application as opposed to its static relationships.
- Exceptions in the processing that have occurred before the application encountered the failure.
- Data that was being processed when the failure occurred.

To make decisions on continued execution, the maintainer needs to know the execution path taken by the system before it reached the point of failure, the possible execution paths of the program/system beyond the point of failure and dependency of an execution path on other paths.

4.2 Execution Model

We propose to address this information need by modeling the execution of the application system. The model describes the system as a network of nodes connected by arcs. Each node represents a system object (program, job, system utility) that processes, collects or consolidates data.

A system normally moves from one node to another on successful completion of the work. A node is marked as successful, partially successful or unsuccessful depending on whether the assigned work of the node is completed, partially completed or incomplete. “Partial Success” refers to the situation where some useful work is achieved without loss of integrity and the system may be allowed to move to the subsequent node. If the system cannot move to the subsequent nodes, the node is termed as unsuccessful. An arc that connects the nodes can either be a data arc or a control arc. Most arcs may have both the roles.

As a large commercial application is likely to have very large number of nodes, the resulting network will be very huge. Building such an execution model for the manufacturing system presented in Section 3.3 would result in tens of thousands of nodes and arcs. It is thus essential that the maintenance team builds the execution model for parts of the application that are prone to failure. They can also build the model at a subsystem level.

4.3 Building the Execution Model

The model is for quick-diagnoses purpose, and our interest is limited to one system life cycle, which is normally a period of 24 hours. Each node has a set of attributes that include a) a timestamp of the beginning of the last execution, and b) a timestamp of the completion of the last execution. The nodes themselves can be sub-classified as follows:

- Begin node, when the system begins the day (at which the date and time are set).
- End node, when processing ends and associated back-ups are taken.
- Dependent node, the one for which one of the mandatory inputs comes from outside the system boundaries.
- Feeder node, the one that generates an output that forms the mandatory input to some other (sub-)system.

The steps in building the execution model for an application system are described below:

1. Collect the list of all the programs, background jobs, utilities (back-up, sort, data reorganization etc.), each a potential node. If a program gets invoked during different times, consider these executions as separate nodes.
2. Create the initial node of the network.
3. Repeat until the final node of the network is reached
   - Add a new node to the network and connect it by data/control arc to the node which must precede it.
4. Identify Dependent and Feeder nodes.

The execution model is a graph that will be used to show the state of system execution with each node reflecting its current status. The existing application programs, transaction controllers and job schedulers can be suitably modified to help us maintain and update the status and timestamp of each node at the beginning and completion of each program. The
execution model can be made readable with a simple coloring scheme (e.g., a successful program as a green node, a partially successful completion as an orange and a failure as a red; nodes currently under execution can be colored blue). The system execution model thus can be displayed as a graph on the maintenance website of an organization to provide a quick snapshot of the day’s status.

5. Usage of System Execution Model

We present use of the system execution model for the example described in Section 3.3. The application is a purchase order processing system in a manufacturing plant. It generates purchase orders based on a material requirements plan and has interfaces to several systems. The system execution model of the relevant portion of the system is presented in Fig 1. The Begin, Dependent, Feeder and End nodes have been identified. Table 2 presents the maintenance requests raised during a period of one week for the same system. Of the total of 25 requests, we have presented a sample of 10 to illustrate the scenarios of how the system execution model can be used in practice. It is possible to mark execution model with the typical paths various business transactions take or parts that execute concurrently. Such static analyses define contexts within which the operational errors can be understood and diagnosed. Some errors (e.g., 5 in Table 3) require debugging at source code level. In such cases, the execution model can point to the offending module for further analysis. A static analysis of the execution model can also help us visualize different error situations and generate rules to guide us in diagnosis. These static analyses primarily consider different ‘what-if’ failure scenarios.

6. Conclusions

In this paper, we have looked at the information requirements for corrective maintenance and concluded that the traditional documentation of system analysis and design may not help the maintainer to fix the problem quickly. To meet needs in maintenance outsourcing with strict SLAs, we have proposed the concept of the system execution model to depict the execution of the system as a graph of nodes and arcs. We have demonstrated how the model can be used in practice by applying it to problem ticket data of an outsourced maintenance project. We find that the model can indeed help reduce the time to debug. It can also reduce the system downtimes by helping the maintainer make quick decisions on further system execution in the event of a failure.

Fig 1. System Execution Model for Purchase Order System

REFERENCES


Table 2. Maintenance Requests for the Material Management System

<table>
<thead>
<tr>
<th>Ticket No</th>
<th>Severity</th>
<th>Time Taken to Fix</th>
<th>Description as Given by User</th>
<th>Analysis of The Problem</th>
<th>Usage of System Execution Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3 days and 2 hours</td>
<td>When attempting to add the new plan, user gets a message, 'two plans cannot exist, please delete the oldest plan'.</td>
<td>When the user enters the details of the plan to be deleted, he gets a message, 'Job is successfully created for Deletion'. But the admin job is not getting created and hence no deletion was done.</td>
<td>In this case, the node &quot;Delete old Material Requirement plan&quot; would be white showing that the node has not yet been executed. Alternatively, the maintainer can check the last completion timestamp of the node and the time stamp will be less than the time of deletion. It signifies that the old plan deletion has not taken place.</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3 hrs 22 minutes</td>
<td>Supplier says he has not received the order.</td>
<td>While the system generates the PO, it also generates the XML message that is transmitted to the supplier. Due to the network problems, the message was not received by the supplier.</td>
<td>Similar to above, the node &quot;Send XML message to supplier&quot; will be in white color.</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5 minutes</td>
<td>Job 1234 failed with a return code of 12.</td>
<td>This job completed successfully but returned with a code of 12 as the previous job which did not have the records to process was terminated soon after initiation.</td>
<td>The maintainer on examination will find that the previous node to this node has terminated with partial success and hence will be in orange. He can then look at the current program/job to see how that status affects the processing.</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1 day and 1 hour</td>
<td>Job terminated with RC 12 - and abend code 1234.</td>
<td>It was found that the design was active in the engineering database but not active in the master design database as the user missed the deadline for upload by a few minutes and was instead of the same.</td>
<td>This is a timing issue. The new design upload will not be a daily job. The begin timestamp of &quot;Provide specification details of prototype&quot; in this case is likely to show a higher value than the time of user upload. The execution model built only for one application will not be useful.</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>6 hrs</td>
<td>User reporting that no reports can be printed.</td>
<td>The reports are printed based on the master data. setup further the user. In this case, there was a mismatch between the assignment of the printer name in the department and CICS definitions.</td>
<td>This is a setup problem and execution model will not be able to help here.</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1 day and 1 hour</td>
<td>User says he is not getting the important stock report.</td>
<td>The printer assigned to the department was down the previous day for few hours and the several printing jobs have been rescheduled for the next day. His job is in the queue waiting for execution.</td>
<td>The maintainer in this case will find that the stock report job is not under execution as the printing jobs is still in color while with an end timestamp that belongs to the previous day.</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>20 minutes</td>
<td>Job 1234 terminated with RC 12.</td>
<td>The job was creating an output sort file which did not have sufficient space on disk.</td>
<td>This problem is similar to serial numbers 1 and 2, and maintainer will be able to answer the question by looking at the time stamp of node &quot;Provide specification details of prototype&quot;</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>30 minutes</td>
<td>Job 1234 failed @ Step 10 or 12.</td>
<td>Lack of master data synchronization between the two systems resulted in the job failure as it was trying to insert the record already present in the table.</td>
<td>This is an environmental issue and the maintainer does not need to use the execution model.</td>
</tr>
</tbody>
</table>