The Effects of Object-Oriented Programming on Hours Spent in Software Maintenance

Kelly M. Turner

Abstract

This paper reports on an empirical study that compares hours spent maintaining software programmed using Object-Oriented design with those using procedural programming. Data was compiled from a location within the IBM service organization. The hours spent per problem were analyzed for 15 components. Individuals supporting these components were also surveyed on their preference of programming styles with respect to degree of difficulty, time spent, and understanding of code flow. This provides support to the results from measurement analysis. The findings from this study generated the hypothesis that Object-Oriented programming requires more time to debug than procedural programming in the Software maintenance cycle.

1. Introduction

Software using object-oriented modeling has become increasingly popular over the most recent years [4], [6], [7], [8]. Industry has been and will continue to be moving towards this technique. Object-oriented design is becoming a dominant programming style and has improved existing designs and products. Operating systems are becoming more and more complex to handle the increasing business needs of customers, thus requiring the enhancements that object-oriented programming can provide. Object-Oriented design is more abstract than previous design methods (in the case of IBM’s z/OS series, PLX and CASE) and gives the designer means to create products that may not have been conceivable prior. Therefore, it is important to study the costs to the maintenance cycle with the increased use of object-oriented design.

Software maintenance is generally the most costly part of the software life cycle [5], [6], [7]. There has been extensive discussion in the areas surrounding software maintenance, much of which has been geared towards developing the best way to support customers while reducing costs. Research involving object-oriented design has looked into methods such as Change Impact Analysis, Program Slicing, as well as procedural and regression testing development and strategizing [1]. Much progress has been made towards technique and process while approaching defects outside of product release.

It is impossible to ensure that large software products are bug-free when they are delivered. Therefore, there is a great need for continued service and maintenance. Software service is an expense to the company if support contracts are offered outside of delivery. With support contracts, the goal is to ensure good customer relations and to maintain current customers as well as to attract new business. Therefore, a smart company will focus on the quality of its support. These are competing statements - Support must be superior to get business but support is only a cost to the company. This introduces the problem of determining the most efficient and effective way to support customers.

Companies maintain records of measurement that gauge how well their support personnel address problems. One important measurement is the hours spent debugging a problem. This measurement is analysed closely and efforts are continually made towards reduction. For companies that have products older than the last 5 years, much of that work has not been influenced by object-oriented design. Therefore, maintenance is still greatly done on procedural programming. Object-oriented modeling influences newly introduced components and/or design. In recent years, software support has been moving towards the support of this design. Have there been any improvements made to the maintenance time since the introduction of the object-oriented model? The arguments favor easier maintenance of object oriented design through better data encapsulation [5], [7]. There has been some evidence in favor of object-oriented programming [7], however the term 'easier to maintain' can be vague. By analyzing the measurements captured, this research will show which programming style (the use of Object-Oriented or not) currently yields the optimum maintenance environment with respect to time spent debugging problems. This time spent can be referred to as turn-around time. Faster turn-around times reduce the cost of software support as well as increase customer satisfaction.

The results obtained from this investigation will give insight into where hours are mostly spent while analyzing defects outside of delivery. We will learn whether the benefits of object-oriented design can also benefit a software company during its maintenance cycle with respect to hours spent debugging problems. This research will help with resource actions by the software company (i.e. Amount of costs expected for resources and employees). Further studies regarding object oriented design and behavior should allow for reduction of this cost. However it is important for current knowledge to understand what the cost is now.

All programming will continue to increase in complexity and great benefit is found in the power behind object-oriented design. It is essential for software support to develop strong object oriented debugging skills to increase productivity as well as customer satisfaction. By researching the hours per problem in these areas it can be determined if there currently is an immediate benefit to the maintenance cycle of software programming in object-oriented design. By finding that object-oriented design causes an increase in the hours per problem measurement, greater focus can be put towards finding ways to reduce this trend.

This paper focuses on a research effort aimed to quantifying the difference between Object-Oriented and procedural programming with respect to the service stream. The results indicate that hours spent debugging problems found in Object-Oriented programs are significantly greater to those debugging procedural programs. In the next section the methodology of the
research is described as well as the analysis of the data. Section 3 details the results of the findings. The next section discusses software support background information and other research efforts in this area. The paper ends with a conclusion to these findings.

2. Research Method

2.1 Obtaining Data

This research intends to clearly show the result of using object-oriented programming when looking at the hours required to resolve a software defect. We will determine with a service provider's current knowledge if introducing object-oriented design reduces problem analysis time. These studies will also show whether maintenance time has steadily increased or decreased over time with both forms of design. This information will provide supporting evidence whether big industry will benefit during the maintenance cycle of software products using object-oriented modeling.

The data used for the research has been obtained by gathering the average hours per problem of 15 components within an IBM operating system over the past three years. Of this group, three use object-oriented programming. It will later be shown that the average hours spent debugging defects is significantly greater for object-oriented components. A small group of 16 software debuggers at IBM have also been polled to give a sense of the current opinion of these two programming techniques, highlighting strengths and weaknesses. This group consists of individuals who have experience debugging problems within object-oriented designed code and code not using object-oriented design.

When asked which programming technique, object-oriented or procedural, is preferred to debug, 13 out of 16 said that they preferred procedural. The software debuggers generally feel that it takes longer to resolve problems found within object-oriented code. The general problem that debuggers site with object-oriented code is that most methods developed to aid debugging code no longer work and there is a need to enhance debugging aids with regards to investigating defects within object-oriented code. This feedback can imply that debugging within imperative code is an area which is more familiar. It also supports the developed hypothesis that Object-Oriented programming requires more time to debug than procedural programming in the Software maintenance cycle.

Object-oriented programming has been used more and more by major companies over the recent years. The products using this programming approach have been introduced to the service stream. For this study, the average hours per problem was obtained on a biannual basis starting in 2000 from 15 components within the z/OS operating system at IBM. The averages were obtained from looking at customer reported problems which were determined to be an IBM defect (labelling the figures gathered as average per defect). Currently there are a much greater number of components within the operating system which do not use object-oriented programming techniques. Therefore, the numbers could be obtained from three components which use object-oriented programming; the other 12 components do not. Of the three components, service data is not available until the second half of 2000. All of the components used are within the same operating system, so they can be compared to each other. One object-oriented component is new at the time the data is gathered, but is shown to have similar averages as the others. There are also a handful of comparably young non-object-oriented components.

The experience of the support providers and the component developers is varied. The development of these components is done by more experienced programmers and decided by the software engineering band level within IBM. Software support providers are at varied skill levels. These skill levels as well as component workload and additional measurements are continually evaluated to ensure the optimal team on each component.

The average per defect was obtained for all 15 components each half year starting in 2000 for a total of 8 units of time. At each timeframe the averages for object-oriented components were compared to the non-object-oriented averages using a simple t-test. The p-value was then obtained using 95% confidence to reject or accept the null hypothesis. The null hypothesis for this research is that there is no difference in the time required to debug using object-oriented programming or procedural programming in the software maintenance cycle. The hypothesis has been tested at each time interval because it is expected that the code complexity within the operating system will be most similar at equal time intervals. This operating system has been upgraded on a half-year basis and each component has undergone some level of enhancement. It is understood that software becomes more complex over time as the product evolves [8].

The hours per problem have also been looked at as a whole for all components to see if there are trends, if the hours spent has been increasing or decreasing. For this study, this will indicate the direction of the trend for both object-oriented components and non-object-oriented components. Predictions of future worth of the object-oriented programming technique can be made. The results of testing this data will show clearly what programming technique (object-oriented vs. procedural) is best for the maintenance cycle presently and in the future.

2.2 Data Analysis

The average hours per problem was gathered for each of the 15 components in each of the 8 time intervals. The problems used for these calculations were customer reported problems which resulted in the discovery of a defect with the product. The actual component names have been changed to CPn where n is between 1 and 15. Appendix A contains the complete data gathered. Figure 1 shows the distribution of the overall mean average hour values for the past 3 years.

Figure 1: Distribution of hours per problem over the past 3 years
A qualitative assessment of the graph indicates that components using Object-Oriented design have higher average hours per problem. This agrees with the hypothesis suggested by the statistical test further described below. For each time interval, a simple t-test was used to test the null hypothesis that there is no difference in average hours per problem between object-oriented components and non-object-oriented components. The alternative hypothesis is that they differ by location of the median. The alternative can be verified by disproving the null hypothesis for the given time interval. For hypothesis testing, the generally accepted significance threshold is $\alpha < 0.05$, which is the value used for this study.

3. Results

3.1 Descriptions of Programs

The programs used for this analysis are within components of the z/OS operating system at IBM. These components are all involved with the internals of the operating system. Half have been a part of the system since the creation; the others have been added as requirements direct. All but one component existed at least 2 years prior to the collection of data. Of the 15 components analyzed 3 were coded with object-oriented design. Other platforms within IBM take larger advantage of object-oriented programming techniques; however the components reviewed for this research include the core areas of the operating system and have the most stable data to collect.

The service providers asked questions regarding their preference in programming technique were all IBM employees who have worked in both components programmed with object-oriented design and without. Sixteen individuals have had experience providing service for at least one each of the two component groups studied. When asked specifically which programming technique, procedural or object-oriented, did they prefer with respect to their position, 13 preferred procedural. Those who preferred object-oriented programming liked the logic flow and use of more meaningful comments. The objective of the object-oriented approach is to simplify testing and maintenance by producing a clean, well-understood design that is easier to test, maintain, and extend than non-object-oriented designs because the object classes provide a natural unit of modularity [5]. However, it has also been discussed that languages programmed with object-oriented design present multiple challenges with program understanding [1]. When asked more specifically which programming technique required more time to debug defects all individuals responded with object-oriented. All service providers involved in the discussion felt that many of the debugging aids developed for procedural programming were not as useful for the components coded with object-oriented techniques. First time failures in the object-oriented components generally result in the need for addition code added for debugging purposes and further recreates. This increases the time required to resolve defects. The information obtained from this discussion gives theories as to why object-oriented components have overall statistically higher hours per problem.

3.2 Differences in Achievement of Service

The median responses for each timeframe of average hours spent debugging defects in object-oriented and procedural components are shown in Table 1 below. The P-Value and Confidence Interval indicate the significance level for rejection of the null hypothesis. Applying a threshold of $\alpha < 0.05$, the results show that most time intervals do not show a significant difference in medians. Three intervals in this timeframe, 12/2001, 6/2002, 12/2002, suggest the hypothesis that object-oriented programming requires more time to debug than procedural programming in the Software maintenance cycle. It is also seen that after 6/2000 object-oriented programming yields a higher median of means. This contributes to the overall hypothesis that over the past 3 years at IBM, object-oriented programming requires more time to debug than procedural programming in the software maintenance cycle. The results from this study suggest the hypothesis that object-oriented programming requires more time to debug in the maintenance cycle. Based on these findings, additional research with additional software companies should further support the hypothesis.

Table 1: Median Hours per Problem analysis over the past 3 years

<table>
<thead>
<tr>
<th>Time Frame</th>
<th>Median Response</th>
<th>P-Value</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-Object-Oriented</td>
<td>Object-Oriented</td>
<td></td>
</tr>
<tr>
<td>6/2000</td>
<td>8.43</td>
<td>6.42</td>
<td>0.204</td>
</tr>
<tr>
<td>12/2000</td>
<td>9.10</td>
<td>11.98</td>
<td>0.385</td>
</tr>
<tr>
<td>6/2001</td>
<td>9.07</td>
<td>10.00</td>
<td>0.723</td>
</tr>
<tr>
<td>12/2001</td>
<td>10.56</td>
<td>16.51</td>
<td>0.002</td>
</tr>
<tr>
<td>6/2002</td>
<td>8.23</td>
<td>13.40</td>
<td>0.020</td>
</tr>
<tr>
<td>12/2002</td>
<td>8.50</td>
<td>13.47</td>
<td>0.023</td>
</tr>
<tr>
<td>6/2003</td>
<td>9.43</td>
<td>14.55</td>
<td>0.060</td>
</tr>
<tr>
<td>12/2003</td>
<td>9.14</td>
<td>11.88</td>
<td>0.166</td>
</tr>
<tr>
<td>Overall</td>
<td>8.41</td>
<td>11.49</td>
<td>0.025</td>
</tr>
</tbody>
</table>
Figure 2 below show the remaining dotplots of the distribution of the mean average hour values for each time interval.

Based on the data, Figure 3 contains a line plot of the average hours per problem for both object-oriented and non-object-oriented components for each time interval.
The trends of hours per problems support the statistical analysis suggesting that object-oriented programming results in increased hours per problem. In appears that the hours per problem has been increasing for object-oriented components, however a larger time interval would be more helpful in determining if this value has leveled off. The hours per problem has remained moderately stable around 8.5 hours for the past 3 years. The line plot also visualizes that object-oriented components have consistently shown considerably higher hours per problem.

4. Discussion

4.1 Software Support Background

The definition of a defect, or bug, is a derivation between the stated specifications of the product and its actual performance [2]. “Odd” behavior from point of view of the customer is not necessarily a defect; it may be determined to be the design of the product. The hours per problem analyzed for this research are against problems determined to be a bug within the given component. This measurement contributes to the two basic categories of engineer’s goals: productivity and customer satisfaction [2]. The concept is to provide the best quality service maintaining the fastest turn-around time (or resolution time). Lower hours per problem allows for faster turn-around time and the ability to assist customers most efficiently increasing satisfaction.

Software maintenance typically accounts for at least 50% of the total lifetime cost of a software system [6]. Software understanding is one of the largest cost factors in software maintenance [6]. Object-oriented design generally is more ‘reader friendly’ however; the concepts surrounding this design make it more complex to debug using the traditional methods used for procedural design.

Resolution time is the amount of time it takes to reach a solution to a problem. In this study the average hours per problem against defects within the given components is being used. This value can vary greatly, from minutes for easy problems and problem rediscoveries to multiple days for more difficult problems. The time it takes to resolve a problem depends mostly on product complexity [2]. Other contributing factors include skill level, workload, and the efficiency of the process and tools used to debug [2]. This study focused on components within the same level of an operating system and a large group of service providers in the same location with similar skill levels using the same process. Tools and service aids will not provide the same effectiveness for both types of programming, procedural and object-oriented [5]. One area for this location to pursue is the current use of service aids and areas in which improvement are most necessary and beneficial. This will benefit the hours per problem measurements for both types of components, where the most needed benefit is with object-oriented programs.

This operating system is divided into components. Component-based software generates better programs by adding stability to smaller applications [9]. The components are independent, task specific, and are allowed to interact with each other. Components are easier to build and debug [9]. This is because the parts can work together seamlessly, and can be developed and maintained by different groups/teams of software engineers. Each component analyzed in this research has been developed by separate teams and have separate teams within one location to provide customer support.

4.2 Related Work

There has been previous research regarding software support as well as maintenance for object-oriented programs. We know that small changes can have major and nonlocal effects in object-oriented designs [1], this complicates debugging. Change impact analysis is used to provide feedback on impacts from any change [1]. Generally, regression test will use this process in environment in which subsets of the changes can be identified and focused on. This is not an easily used process with current service aids when debugging customer reported problems. A greater area of the component code could be involved when defects are detected in object-oriented code due to the unique aspects of object-oriented design. Traditional debugging methods do not direct themselves to notions such as classes, inheritance, and encapsulation [7]. Fault-proneness metrics allow for fault-proneness detection within object-oriented products [3]. These metrics have been defined and validated for various goals including reusability and maintenance [3]. Research is continuing to refine methods exploiting fault-proneness metrics.

The standard maintenance problems exist with the introduction of object-oriented design, as well as new problems introduced by the addition of objects [5]. The common tools used to aid with service of components generally provide no direct benefit to the maintenance process of object-oriented design. Knowledge tools, which focus more on the thought process to determine what the code and changes to the code should be doing, are the most beneficial when looking at object-oriented design [5]. Continued efforts are being made to enhance the serviceability of object-oriented components.

5. Conclusion

We investigated the maintenance costs of supporting object-oriented and non-object-oriented source code. This cost is factored in terms of personnel hours spent debugging a problem. The findings provide support to the original hypothesis that object-oriented design increases the hours spent debugging problems found in the maintenance cycle. Further research and data encapsulation is expected to support these initial findings. Much research has currently gone into default analysis of
object-oriented design. The knowledge obtained from these studies and future studies will help better support. Focus should be moved towards education for support organizations. This education should introduce newer programming styles and techniques as they become available and are exploited by the developers. It should also teach support personnel various methods which help debugging intricate problems. Our goal is to reverse the conclusions that have been made to increase the effectiveness of object-oriented programming across all stages of the software lifecycle.

References


