Designing SUPPORTABILITY into Software

by

Prashant A. Shirolkar

Master of Science in Computer Science and Engineering (1998)
University of Texas at Arlington

Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management
at the
Massachusetts Institute of Technology
November 2003
[February 2004]

Signature of Author _____

Prashant Shirolkar
System Design and Management Program
February 2002

Certified by _____

Michael A. Cusumano
Thesis Supervisor
SMR Distinguished Professor of Management

Accepted by _____

Thomas J. Allen
Co-Director, LFM/SDM
Howard W. Johnson Professor of Management

Accepted by _____

David Simchi-Levi
Co-Director, LFM/SDM
Professor of Engineering Systems

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Michael Cusumano for all his encouragement support, and motivation throughout this 'distance learning' endeavor. The immense faith he bestowed upon me encouraged me to tackle this large problem. I would also like to thank the SDM Program for giving me the opportunity to perform research on a topic that has growing implications in the software industry and very close to my heart.

Next I would like to thank Kevin Collins, Jamie Burroughs, Ramkumar Pichai, Ashvin Sanghvi, Alan Padula, Robert Dorr, Bob Ward, Jeremy Phelps and Lori Turner for their invaluable time, patience and without whom I would have had no data or information for this thesis. A special mention goes to my supervisor, Stephen Pepitone, for his active participation in our numerous discussions, constant guidance, encouragement and proof reading, all of which were critical towards the completion of this thesis.

I would also like to thank my parents and sister for their support and patience. Lastly but not in the least I would like thank my wife, Kateryna for her unconditional love, support and understanding despite her being pregnant and without whom this thesis would still have remained a distant dream.

1. INTRODUCTION

1.1. Background

Despite the identification of a "software crisis" in the 1960s, large software projects are full of defects even today [19]. These defects cause the software to behave unexpectedly. The cause generally is attributed to the inherent property of software to possess arbitrary complexity [20]. Additionally, computer software is becoming all pervading. There is literally an explosion of software automating things that were considered impossible to automate only a few years ago. Essentially, software is growing to encompass anything and everything with such speed that we don't have the time to reflect on 'what happens when the software does something that it was not intended to do?' In other words, the software encounters an anomaly or a defect something customers encounter everyday.

In any case, whenever customers encounter problems caused by defects they need help to solve these problems. They generally seek such help at a software vendor's product support services. With software getting more and more complex it is getting extremely difficult to support customers in a timely and cost effective manner. There is a constant struggle to ensure high Quality of Service (QoS) with low Total Cost of Ownership (TCO), which in turn affects Return on Investment; in such a scenario considering nearly 80% of the costs associated with software occur after it is shipped to the customer. Currently, there are two solutions to such problems caused by defects:

First, if the source of the problem itself is found to be a defect in the software then the software vendor generally fixes the problem by modifying their code. Such a problem is generally classified as a bug. By now it is known that software applications have such bugs or defects or anomalies. These bugs cause the software to behave unexpectedly. Depending on its severity this behavior can on one extreme go unnoticed or one the other end produce catastrophic results. It pervades all software whether it is the AT&T long distance switching node crash in 01/15/90, the well known Y2K bug in database related software, Microsoft Passport Security defect, Mars Polar Lander defect, or the Ariane 5 rocket defect. These are just a few examples of well known software bugs. For each well known bug there are untold numbers of bugs that may not see the limelight. These defects can arise due to many reasons during the design and development of the software however they primarily arise due the essence of software to possess inherent properties of complexity, conformity, changeability, and invisibility [20]. Hence, a question arises, "Do we still try to single-mindedly go for the elusory 0 bug product? OR Do we accept that there are going to be bugs and try incorporate / accommodate for this directly into the product or otherwise?"

Second, if the source of the problem is such that the vendor cannot simply fix the product, then it is classified as a limitation of the software product itself. In such a case the support organization works with the customer to find a workaround. Additionally, if many customers encounter the same problem then the product needs to be modified based on the workarounds their support services have provided to customers. In other words the product evolves based on how its customers use it.

Product support services also receive numerous problems that are non-defect related where the source could be customer error or ignorance, improper documentation etc and their solutions generally require end-user education and training, better documentation etc. However such non-defect related support incidences fall outside the scope of this thesis.

In the defect related scenarios illustrated above most of the contemporary efforts to determine the root cause of a customer's problem, requires in-depth knowledge of the product, extensive debugging that is extremely tedious, painstaking, and manual. Resolution of a problem depends solely on the troubleshooting skills, experience and subjectivity of the Support Personal that the customer is able to get a hold of. Products have all along incorporated functionality such as better features, performance, scalability etc. into their designs however none explicitly addresses how to tackle a software application's problems. This is unfortunate since such problems deeply affect customers and software development organization alike. The former looses productivity, effectiveness, time and money while the latter expends dedicated monetary and human resources for troubleshooting software products for a number of years until the product is declared obsolete or the customer's license expires. Problems affect overall customer satisfaction with the software making them more skeptical of future releases. The bottom-line is that there is a strong need to innovate software design and development itself to address how to deal with software problems so that customers should not have to loose their time, money or their life because the software application they are using has problems.

1.2. Objectives

The major objective of my thesis is to study and explore how software applications should directly or indirectly incorporate for this ability of software to have defects so when a customer encounters a problem it an be resolved relatively fast without sacrificing overall customer experience. This entails addressing the question posed earlier that we need to accept that there are going to be defects and try to accommodate for this directly into the product or otherwise. The basic problem space to address this question can be broken down into the following questions:

- Why should software incorporate supportability into the design?
- What should software incorporate in order to design or accommodate supportability into software?
- How should software incorporate in order to design or accommodate supportability into software? This entails asking how much supportability needs to be incorporated into the software.

1.3. Methods and Approach

This thesis is a conglomeration many of the disciplines taught in the System Design and Management Program. First, this thesis draws extensively upon information learnt in Software Engineering. Next it incorporates principles and techniques from Dynamics of Innovation to determine why supportability is needed. Additionally concepts learnt in Systems Engineering,

Systems Architecture and Organizational Processes are used to provide a holistic approach to tackle how supportability can be incorporated into software. Lastly but not in the least it draws from Lean principles to recognize the need to eliminate defects.

In analyzing the current supportability initiatives this thesis incorporates concepts from both engineering and management. The engineering side involved a detailed analysis of the supportability initiatives of four software products at Microsoft. This included mining data on support incidences over the past year to answer the 'What?', 'Why?' and 'How?' and determining effectiveness of some of the initiatives being undertaken. Additionally, using this information to determine a holistic framework draws from Software Engineering, Systems Engineering and Architecture. The management side involved determining the need for software innovation draws from product innovation principles studied in Disruptive Technology, recognizing the muda [40] in supporting contemporary software, and understanding that there are strategic and cultural issues in addition to technical issues to ensure that supportability is incorporated into software products successfully.

Gathering information for the thesis comprised conducting interviews with Microsoft Personal, as well as those from other commercial software firms, analyzing Microsoft Products related defect databases, and researching software journals such as IBM systems Journal, Communications of the ACM, ACM Transactions in software Engineering, IEEE Software, IEEE Conference in Software Maintenance etc. over the past fifteen years.

1.4. Structure of Thesis

The next section covers a literary search that clarifies what is supportability in software as well as what part of supportability this thesis concentrates on. Initial steps would include understanding various aspects of supportability contemporary software products entail. This entails understanding and getting abreast with the research being carried out in this aspect.

Next we analyze information concerning the 'What?', 'Why?' and 'How?' of ongoing supportability measures concerning four Microsoft software products, which will include client-side and server-side software products. These products are Microsoft Windows, Microsoft Office, Microsoft SQL Server and Microsoft Internet Information Server. This includes mining data from all the support incidences received over the past year concerning these four products, researching defect related databases to understand how supportability issues are prioritized for inclusion into the software design, effectiveness of some the features already incorporated into software, and interviewing numerous Microsoft Personnel that are currently working on various supportability initiatives concerning these four products.

Next we also analyze supportability related initiatives at IBM and a prominent software company that designs and develops printers. Interviews with personnel from other companies, who are involved with software supportability, are also being considered. The next section assimilates a

framework that facilitates incorporating supportability into software based on information from the industry and principles gleaned from the SDM curriculum. This essentially entails recommendations as to how to incorporate supportability into software. Finally we elaborate what the future holds in this field of research.

2. LITERATURE REVIEW

2.1. What is software supportability?

When software running at a customer site deviates from its intended or normal behavior in such a way that the customer needs help in order to overcome it, then there is a need to support the software. Such a deviation is caused due to inherent complexity in software, problems during any or all of the phases the software's development life cycle, incorrect customer training for operating the software, customer Error.

The ability of software to diagnose and correct such an eventuality either intrinsically in its design or extrinsically through support tools and / or human intervention broadly comprises software supportability. Supportability issues includes maintainability, training, human factors, reliability etc. [5].

2.2. Why is there a need for supportability in software Systems?

According to Osterweil [4], when a piece of software does not perform as required and expected, a software failure occurs. Such a software failure is colloquially referred to as a bug. It is well known by now that software systems have bugs. Depending on its severity this deviant behavior at its extremities can either go unnoticed or produce catastrophic results. It pervades all software

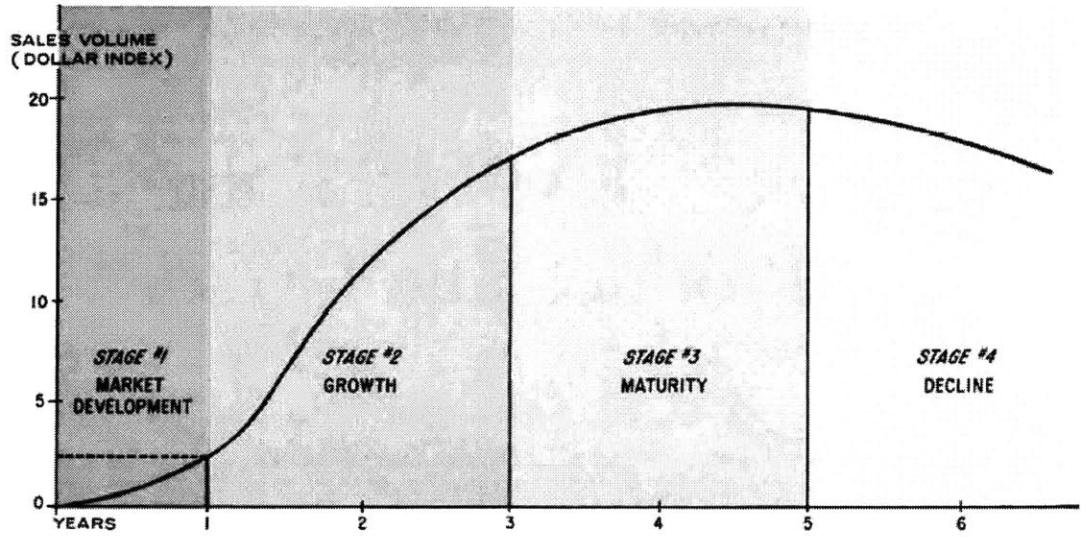
whether it is the AT&T long distance switching node crash of January 15 1990, the Y2K bug in database related software, Microsoft Passport Security bug or the Ariane 5 rocket bug. These are just a few examples of well known software bugs. For each well known bug there are untold numbers of bugs that do not see the limelight. Hence, a question arises, "Do we still try to single-mindedly go for the elusory 0 failure product OR accept that there are going to be supportability issues and try to incorporate it into the product?"

Products have incorporated functionality such as better features, performance, scalability etc. into their designs however very few explicitly account for supportability. This is unfortunate since it directly affect customers and software development organization alike. In general, bugs affect customer productivity, effectiveness, etc. and large sums of money, resources and time are spent supporting / servicing software products for a number of years until the product is declared obsolete or the customer's support license expires. According to Herrin [7], the cost of maintenance is a substantial fraction of the total life cycle cost of a software system. A study estimated that 60% of all business expenditures on computing were for maintenance for software written in COBOL [6]. The more complex the system the more expensive it is to maintain [3]. This also affects overall customer satisfaction with the product making them more skeptical of future releases. Hence, supportability should be one of the key requirements for developing a software product. The customer should not have to loose their time, money or loose their life because the software system they use has supportability issues.

2.3. Software Product Maturity

In general every successful product follows a market lifecycle pattern which generally includes multiple stages, starting with when it is newly introduced into the market and tries to gain a foothold, followed by an increase in its popularity as more customers buy it as well as more competitors try to copy its success, next its popularity and market somewhat stabilizes and finally it starts to decline. Theodore Levitt [15] lists these stages as Development, Growth, Maturity, and decline.

EXHIBIT 1. PRODUCT LIFE CYCLE — ENTIRE INDUSTRY

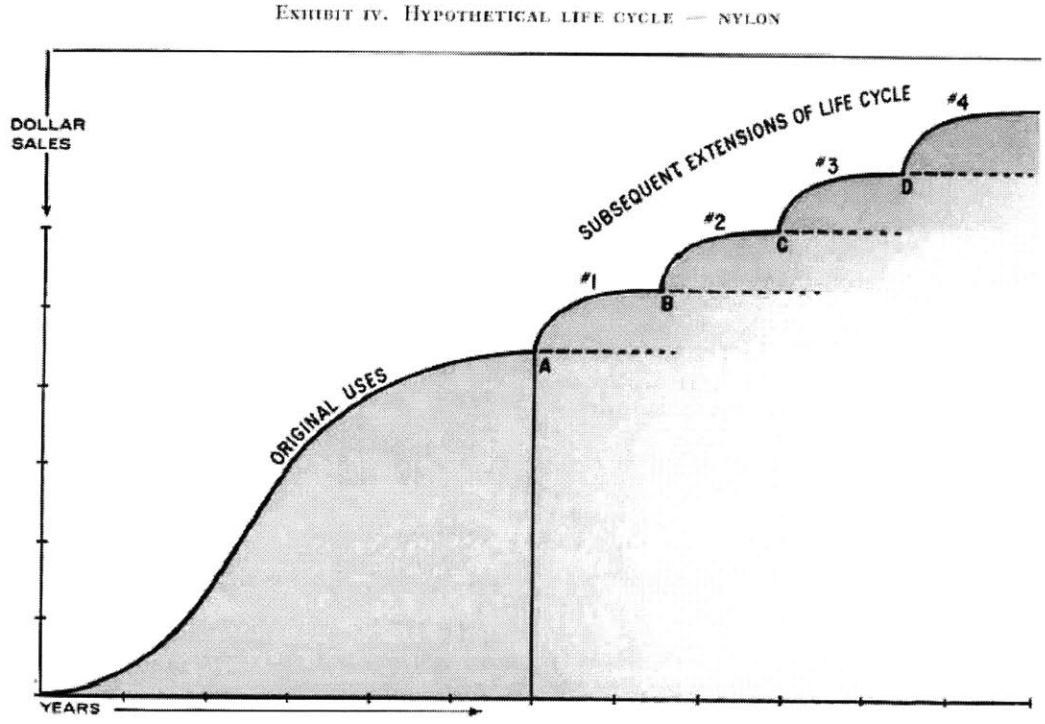


Source: "EXPLOIT the Product Life Cycle"; Theodore Levitt; Harvard Business Review; Nov/Dec65, Vol. 43 Issue 6, p81, 14p

Figure 1: Generic Product Life Cycle

There are modifications to this lifecycle depending on the product. For example a product can continuously reinvent itself by developing newer uses like nylon. Each new use spawned

essentially a new market for the same product. This in general increases the lifecycle of the main product itself, which in this case is nylon. To a large extent the software lifecycle seems to draw similarities from this type of lifecycle. Every new version of the software seems to try to up the ante by striving fervently to provide newer feature sets with the hope of providing more usefulness to the end user.



Source: "EXPLOIT the Product Life Cycle"; Theodore Levitt; Harvard Business Review; Nov/Dec65, Vol. 43 Issue 6, p81, 14p

Figure 2: Modified Product Life Cycle

2.4. Total Cost of Ownership (TCO)

This constant cycle of reinventing the software product every few years tends to lay major emphasis more on developing better and more numerous useful and appealing features each time than how well can the product recover when a problem occurs. Without doubt the software quality has improved over time however with the continued emphasis on writing more features, more complexity and developing it even faster than before leaves very little (if any) emphasis on supportability aspects of the software. In addition to these constraints there is a need to continuously reduce the cost of supporting the software on the part of the software organization that designs and develops it and the cost of using the software on the part of end user. This is the Total Cost of Ownership. The way successful software is currently being designed and developed is resulting in a spiraling higher Total Cost of Ownership for the customer as well as the software developer organizations and here is why:

On the customer side their expectations of software products in general are much higher than a few years back. For example, during the Developing stage of the lifecycle of the Windows 95 Operating System, customers in general viewed the infamous 'Blue-screen-of-death' differently than during the development stage in the lifecycle of the Windows XP operating system. They expect such occurrences to be less frequent (if at all) than before. Additionally, they expect the software to recover more gracefully from such a predicament because firstly, this is not a new occurrence and secondly, any other product, including software that provides value to a customer is expected to evolve by learning from its mistakes and making sure they do not recur. This

natural evolution reflects the fact that after using the software product, its customers expect a certain quality from the software core irrespective of the added feature set offered by newer versions. With every new version this quality is expected to increase. Unfortunately, for software the evolution cycle has been accelerated many times over as compared to any of its predecessor products in the industry.

In addition to this expectation lies the matter of dependence. Computers in general have replaced or augmented a host of tasks in various fields in which customers were used to depending on humans. These tasks include diagnosis in medicine, medicine prescription in pharmacy, monetary transactions in banking and accounting, creating professional documents and presentation, communications, etc. Hence, customers tend to rely a lot more on computers than they did even a few years back. As a result, the cause and effect relationship between customers and software has a much more far reaching consequence than before. For example if a problem occurs with a particular computer depending on its prevalent use it can partially or completely paralyze a single person's day-to-day activity or that of an entire network. This inherently increases their Total cost of ownership in the software because they have invested a lot in terms on money and time on the software and expect the software and software organization to reciprocate likewise.

This in turn has far reaching effects on the organization that designs, develops and sells the software as well. When a successful software product is in the development stage, customers are still trying to realize what value it affords them, hence their expectations are low. With increase in time as each customers buy-in to the value provided by the new software during the growth

phase their expectation increases. The software organization now needs to meet this expectation by creating an infrastructure that helps ensure that the software meets customer expectation. This infrastructure gives rise to the "Customer Support" infrastructure organization and the "Customer Satisfaction" metric used by most organizations if not all to measure their performance of their respective support infrastructures. The effectiveness of this infrastructure to meet expectations also depends on the size of the customer base and the base each customer represents. Hence, the expectations that the organization needs to shoulder depend on the size of the customer base, the time they have spent on it to realize its value and the resources they have invested in it to incorporate the value offered by the software into their value stream. Additionally, this support infrastructure needs to also shoulder the responsibility of not just supporting a newly released version of the product but each prior 'reinvented' version of the software.

Depending on how much value the customer tries to extract from the software the total cost of ownership changes form both the customer and the organization the makes the software. Hence successful software that have provided good value for the customer and have been around for some time inherently have a higher Total Cost of Ownership. In such cases it becomes even more imperative for organizations that ship them to concentrate more on the improving supportability aspects right from the conception of the future software versions. This will afford them the ability to control their share of the Total Cost of Ownership once their product ships.

Over time improving the supportability aspects of a stable / mature product involves a learning process that makes understanding the customer easier and reducing R&D product lifecycle costs. This can be applied to software running on contemporary HP Printers that provide better printing quality, speed and error detection and recovery than those a few years ago. Gone are the days of meaningless errors such as "LPT Error". Nowadays printers not only output the error, they also convey the cause and how a user can address the cause. For example, in the scenario that a paper jams, the printer displays the error, then proceeds to display where that paper is jammed with steps that should be taken to remove it. This eliminates the need to call the printer specialist or HP product support for a paper jam. Another factor that helps reduce maintainability TCO costs for the organization is the changing of the 'Skunkworks' culture of R&D. Nowadays when a project ends a few engineers from the project team devote all or part of their time on code maintenance. In addition, techniques to involve the customer earlier in the development cycle and getting their feedback using Beta Product Releases and Focus groups, where customers give feedback on usability, supportability, deployment etc. are being used to reduce supportability associated costs once the product ships to the customer.

2.5. Quality of Service (QoS)

Despite the continuous evolution and increased complexity of software in general, Product Support Services are constantly striving to improve a customer's overall experience while operating the software. This is also called the Quality of Service (QoS) rendered to the customer. This is a gauge of supportability of the software product. It is a constant challenge to improve the Quality of Service while trying to reduce the Total Cost of Ownership.

There are a couple of metrics that help gauge software supportability - Customer Satisfaction and Days to Solution. Customer Satisfaction, as the name implies measures how satisfied the customer is with the manner is which their software problem dealing was supported. Days to Solution (DTS) indicates the number of days required to solve the customer's problem. In general both these two factors are tightly inter-linked to Quality of Service in that if the customer is dissatisfied and the time taken to solve their problems is high then Quality of Service needs to be improved. In turn, bad QoS results in higher total cost of ownership for both the customer as well as the Product Support Organization. Hence, in general

$QoS = f(CS, DTS)$

Additionally, Days to Solution and Customer Satisfaction are inversely proportional. In other words the faster a problem is resolved more satisfied a customer is and vice versa. Hence,

$CS \alpha 1/DTS$

2.6. Supportability versus Maintainability

In general, that there two feedback loops to R&D or the Design and Development team from the customer. The first loop feedback occurs when a customer encounters a problem that requires fixing on the software code. This loop generally falls under the aegis of maintainability. More precisely, IEEE defines software maintainability as "Modification of a software product after

delivery to correct faults, to improve performance, or other attributes, or to adapt the product to a changed environment." [21]

The second feedback loop occurs when a customer encounters a problem that gets resolved by offering a workaround. Such a collection of workarounds can form a list of best practices which is conveyed back to the Design and Development Team. Additionally this loop could comprise a wish–list of changes to improve the product as a whole. This falls under the broader umbrella of supportability and it is crucial to consider for future major and service releases. Strategically it can also be considered a preventive maintenance. Another activity that solely falls under supportability is training the support professionals at the call centers so that they can help customers when they call in with problems, an aspect of supportability that falls outside the scope of this thesis.

Broadly speaking, the Air Force Operational Test and Evaluation Center (AFOTEC) supportability model [22] evaluates supportability to encompass:

- Maintainability factors such Documentation, Source code and Implementation.
- Software lifecycle process factors such as project management and configuration management.
- Software support resources such as support personnel, support systems and support facilities.

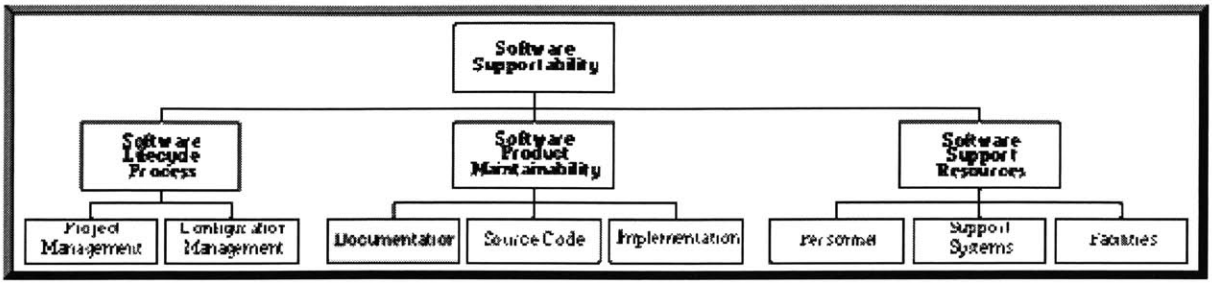


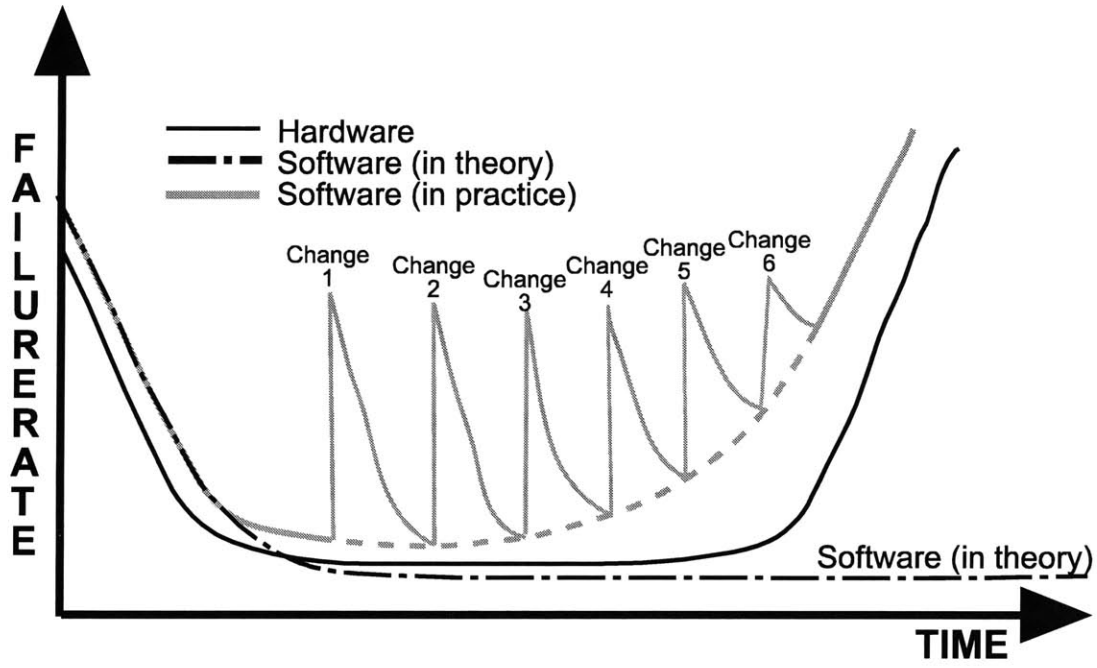
Figure 3: AFOTEC software supportability model.

2.7. Software Characteristic

Hardware wears out with time; however software does not wear out with time [10]. The failure rate of hardware can be plotted to depict this wear out. Initially, hardware often has a high failure rate early in its life (defects per unit time) until manufacturing or design defects are corrected. The failure rate then drops to a steady-state level for some period of time until it begins to wear out due to cumulative effects of environmental factors such as dust, vibration, abuse, temperature extremes etc. At this point, the failure rate begins to climb again. Plotting the failure rate against time results in a relationship often termed as the "bathtub curve".

Unlike hardware, software is not prone to environmental factors and hence theoretically does not wear out. Hence, once again in theory it should maintain steady-state till infinity. However, in practice this does not happen. Software is found to deteriorate with time and changes. As changes are made, new defects are introduced and latent errors are uncovered causing the failure rate to spike, until the new defects or latent errors are diagnosed and fixed. This spike in the

failure rate is noticed each time new change is incorporated into software making the cumulative software failure rate curve steadily increase. All the three types of curves are illustrated in the figure below. This characteristic of software change profoundly affects customer QOS and overall software supportability TCO.



Source: "Guidelines for Successful Acquisition and Management of Software Intensive Systems", Chapter 11, Version 2.0 June 1996.

Figure 4: Bathtub curves for software and hardware

2.8. The problem space

The problem space essentially entails software systems that incorporate supportability in one way or another and those that do not. The objective of this thesis is to study and explore software systems that can incorporate and have incorporated supportability. Software systems that incorporate supportability in form or the other can be classified as follows:

- **Manual supportability**: Systems depend on human intervention to diagnose and correct an anomaly.
- **Semi-Automated supportability**: Systems that are automated to diagnose and sometimes correct some anomalies however need human intervention to diagnose and correct all possible anomalies. This intervention could be in the form of the system helping the customer determine and fix the problem themselves or the customer has to resort to external help after narrowing down the problem to some extent after leveraging the self-help capability of the system.
- **Automated supportability**: Systems that do not depend on any human intervention to diagnose and correct with anomalies. These are self-supporting software systems.

3. SUPPORTABILITY AT MICROSOFT

This chapter covers Microsoft's initiative on supportability. Here we will be observing the different approaches to supporting client and server software applications. Depending on whether they are client or server applications customer requirements and expectations differ hence, Microsoft software applications in both categories are presented. These software applications include on the one hand Microsoft Windows Operating System, Microsoft SQL Server and Microsoft IIS Server software application with a server perspective and the Microsoft Office software application with an predominantly client perspective.

Data was obtained from interviews with the respective product group representatives, existing product documentation in the form of on-line help, knowledge base articles and support related data, as well as trying out the supportability features in a real-world environment. In addition to these software applications I will also include discussion on software currently being developed at Microsoft that is geared towards better supportability. These include the Microsoft Operations Management (MOM), Bayesian Network Editor and Tool Kit, Checkpoint Troubleshooter, and Detours.

3.1. Microsoft Windows Operating System

Microsoft Windows is the operating system developed by Microsoft. Depending on its usage there are different versions of the Windows Operating system. If single users at home or in the

Office are generally going to use the operating system then for them there are the Windows 95, Windows 98, Windows ME and Windows XP versions of the operating system. On the other hand, if the operating system is generally expected to serve multiple users then there are the Windows NT, Windows 2000 and Windows 2003 versions of the operating system.

3.1.1. How was supportability tackled?

This objective came about after analyzing numerous incidents or issues or problems that customers called into Microsoft Product Support Services with regards to the Windows 2000 and Windows 2003 operating systems. Each technological area was ranked on the basis of how many incidents were received, and the amount of resources related to labor that were expended on it. The next 2 tables list the major issues encountered with the Windows 2000 and Windows 2003 Operating Systems. The prioritization is based on a variable called the **Supportability Factor (SF)** calculated as a weighted measure of the number of Issues (I) received, and the amount of resources (R) expended on the trying to resolve the issues for a particular component of the Windows 2000 / 2003 operating system. Hence, for the nth component of Windows its Supportability Factor (SF) can be calculated as:

$\mathbf{SF_n = f\,(\Sigma I_n, \Sigma R_n)}$, for the **nth** component of the product.

Rank	Component	SF

1	Domain Naming Services	1.10
2	Application Compatibility	0.61
3	Clustered Servers	0.46
4	Networking	0.29
5	Faulty Hardware	0.23
6	Kernel-HAL	0.21
7	Hardware Abstraction Layer	0.15
8	Security Policy	0.13
9	Terminal Server	0.11
10	Authorization	0.10

Table 1: Windows 2000 supportability Components based on SF.

Rank	Component	SF
1	Domain Naming Service	1.75
2	Networking 3rd party hardware	0.39
3	Application Compatibility	0.30
4	Security Policy	0.29
5	Active Directory Replication	0.22
6	Terminal Service Licensing	0.21
7	Authorization	0.16
8	Faulty Hardware	0.11

9	Operating System Drivers	0.07
10	Active Directory Review / Deployment	0.05

Table 2: Windows 2003 supportability Components based on SF.

Based on this information each area of expertise was comprehensively analyzed by the Design and Development teams and Product Support teams to come with a list of recommendations to improve supportability of each technology area as well as an overall objective was determined.

3.1.2. Supportability Objective

The supportability objective for Windows is two-fold. The first objective is empower customers so that they can help themselves solve their own problems by way of better diagnostic and help capability. In other words, they do not need to turn to assisted support. Second, even if they do have to resort to assisted support then the need is to ensure that they can do so properly. Such a scenario would consist of a step by step process on contacting support and being informed whether support is making progress towards solving their problem or not and if so then how long would it take support to solve their problem.

The first objective stems from the intent to reduce the rework or waste resulting from repeated round-trips or exchanges between the support professional and the customer to get all the relevant data in order to solve the problem. This is being achieved by improving diagnostic data.

In order to obtain better diagnostic data dramatically requires improved detection capability in the form of better error reporting, data contained in core dumps during a crash etc.

This essentially entailed improvements to the 'Dr. Watson' tool, which been the primary data gathering tool when a crash occurred on a Windows Operating System. Dr. Watson for Windows is essentially a program error debugger. The information obtained and logged by Dr. Watson is the information needed by technical support groups to diagnose a program error for a computer running Windows. A text file (Drwtsn32.log) is created whenever an error is detected, and can be delivered to support personnel. It also provides an option of creating a crash dump file, which is a binary file that a programmer can load into a debugger. If a program error occurs, Dr. Watson will start automatically. This tool possesses the following enhancements:
Provides Error and Event alerts with improved local self-help,
Provides the capability to upload error information to Microsoft if the customer so desires. Error Information ranges from simple information about the product that caused the error to core dumps in case of the application crashing.

The enhanced Dr. Watson tool allows the customer to upload the dumps and error reports via an Error reporting tool (covered in the Microsoft Office sub-section 3.2) to a centralized Microsoft Online Crash Analysis [17] Server, which has the capability to analyze the crash dump or the error report and to determine what type of issue it is and whether a solution exists to the problem or not. The figure below describes the status of a crash analysis for a problem I reported. This website also allows a customer to track their problem on an on-going basis and delete any reports that have been resolved.

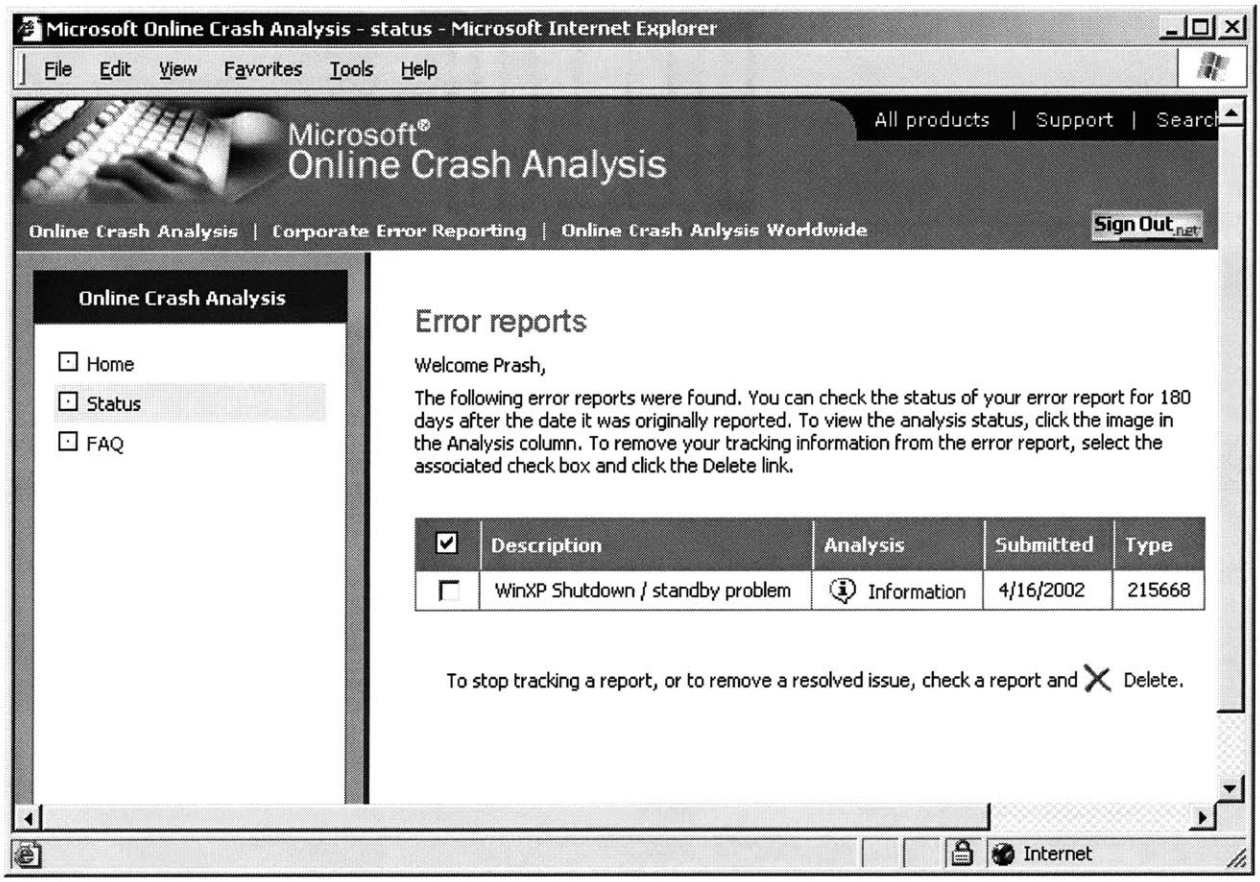


Figure 5: Status of a Dr. Watson Online Crash Analysis

The next figure below depicts more detailed information on the problem I reported at the Online Crash Analysis website. This error report identifies that the problem is in a device driver installed on my system, however the cause has not yet been determined yet. Additionally it also provides more information as to what could be causing to the problem as well as a means to empower the customers to troubleshoot the problem further on their own in a more informed manner by way of knowledge base articles and helpful links.

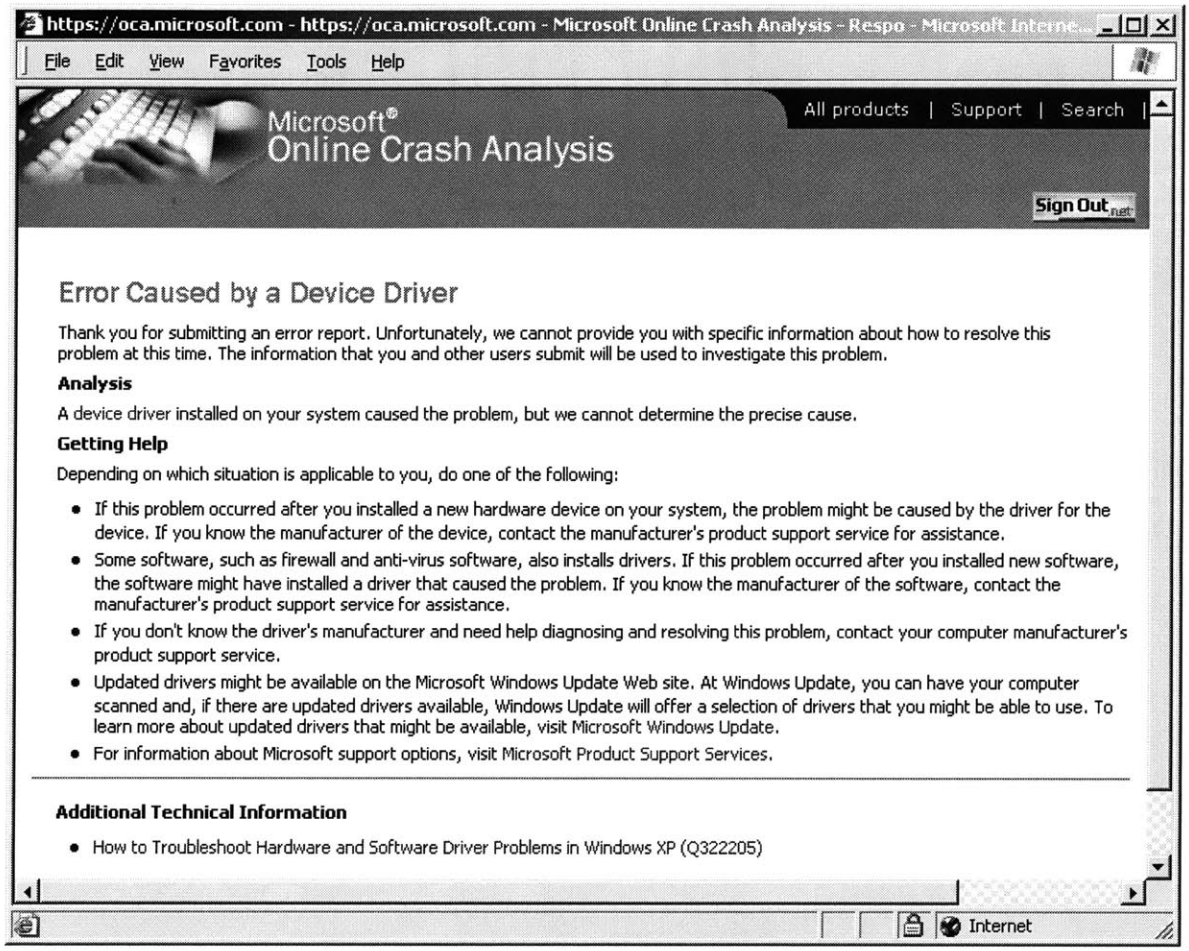


Figure 6: More Information on the Error report

This tool has been extremely effective in compartmentalizing all the different problem 'buckets' or segments that Microsoft customers encounter in the entire problem space. This helps in determining possible problem areas and trends in the way customers are using Microsoft software. Another, advantage of this tool is that is provides an effective means of finding the problems customers are currently facing and in conjunction with the Microsoft Windows Update website appropriate solutions to their problems can be deployed.

The second step in achieving the supportability objective is to assist customers still cannot solve their problems and need to reach out to Product Support services for help. In such a scenario customers are provided with the self-help component. This component runs in tandem with an Error Reporting Component. The primary objective of the self-help component comprises troubleshooting the customer's problem so that they can take a more informed next step that will help in getting to the root cause of their problem. It resides locally on the customer's machine similar to the error reporting tool. It can analyze dumps as well as automatically communicate with Microsoft's online information database on customer issues and suggest the cause as well as steps to resolve the problem.

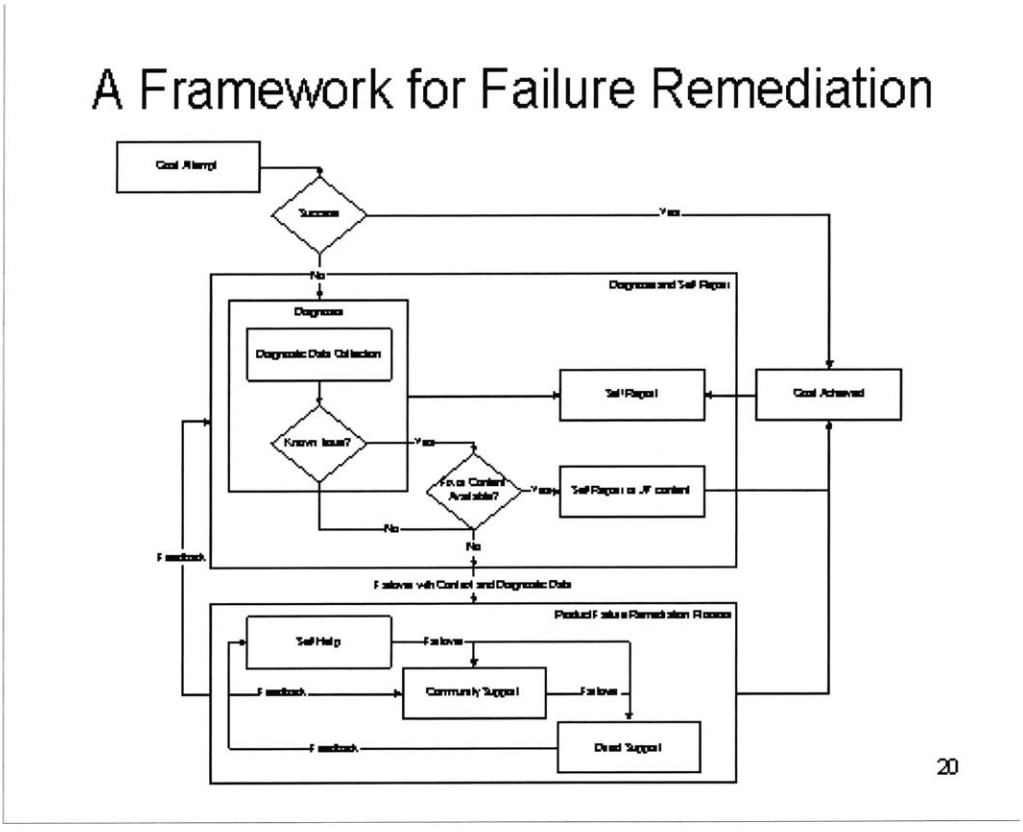


Figure 7: A Framework for Failure Remediation

This general framework illustrates at a high level the ability of the self-help component to determine the amount of help it can assist the customer with to take a more informed decision towards solving their problem. The self-help capability is bolstered by an Online Help and Support Center. The first of its kind was introduced with the Windows XP operating system and represents a significant milestone in delivering a single resource for Online Help, support, tools, how to articles, and other resources. Extensive Online Help is accessible via Search, the Index, or the table of contents. Additionally, it is easy to get Help from an online Microsoft support professional, trade questions and answers with other Windows XP users on Windows newsgroups, or use Remote Assistance to have a friend, co-worker, or Helpdesk professional assist you.

3.1.3. Recovery and Restore

An important component of supportability lies in the ability of the product to recover after the problem has occurred. Windows XP introduces Automated System Recovery (ASR), an advanced option of the Backup tool (NTBackup.exe). ASR provides the ability to save and restore applications, the system state, and critical files on the system and boot partitions. ASR replaces the Emergency Repair Disk used with Windows 2000 and Windows NT 4.0. This feature also provides the Plug and Play mechanism required by ASR to back up Plug and Play portions of the registry and restore that information to the registry. This is useful in a variety of disaster recovery scenarios—for example, if a hard disk fails and loses all configuration parameters and information, ASR can be applied and the backup of the system data is restored.

The Windows operating system from Windows XP onwards provide the capability to save a snapshot or a checkpoint of the system, called the System Restore [38]. System Restore is a feature first introduced with Windows Me that automatically monitors and records key system changes on your computer. When a customer makes system changes that may have caused problems, they can undo the change or even revert to a configuration that existed when the system was performing optimally.

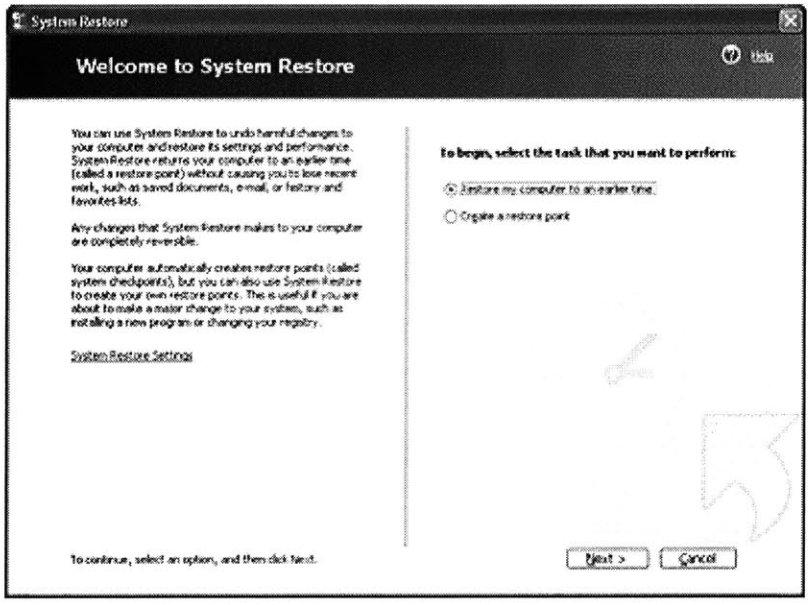


Figure 8: System Restore Checkpoint Capability

System Restore does not make changes to personal data files such as documents, drawings, or e-mail. It actively monitors changes to the system primarily changes to the registry and to certain application files, and automatically creates easily identifiable restore points. Windows XP creates these restore points once each day by default, as well as at the time of significant system

events such as application or device driver installation. You can also create and name your own restore points at any time. System Restore does not monitor changes to or recover your personal data files. The objective is to empower customers to keep their computer running smoothly without the need to seek help from other sources.

3.2. Microsoft Office

Microsoft Office is a client-side software suite or package of software solutions developed by Microsoft to empower customer to better perform their day to day office functions. Each component of the suite targets a particular office function such as word processing, spreadsheets, presentations, project management, emails, database management etc. Examples of Office components are Word, Excel, PowerPoint, Project, Outlook, Access etc. The current version of Office is Office 2003.

3.2.1. Supportability Objective

Supportability concerns in Office 2003 primarily stem from improving the overall end-user experience. In order to achieve this goal they concentrate on graceful Application Recovery and enhanced Error Reporting. This section delineates these enhancements.

3.2.2. Reactive supportability Features

Error Detection and Troubleshooting for Office 2003

The Dr. Watson tool provides the enhanced Error Reporting functionality first introduced in Office XP. The information now reported back to Microsoft and access to resolutions is further enhanced. A new component called the Service Quality Monitoring (SQM) has also been added

to gather data in order to improve the users experience in Office 2003 and future editions of Office. Here is an example of how Error Reporting works in Office 2003:

The initial Error Reporting dialog-box informs the customer that an unexpected problem has occurred and the error reports are being uploaded to the Microsoft on-line crash analysis server. The user has the choice to send the data immediately or wait for a later more opportune time. Additionally, they check for more details as well.

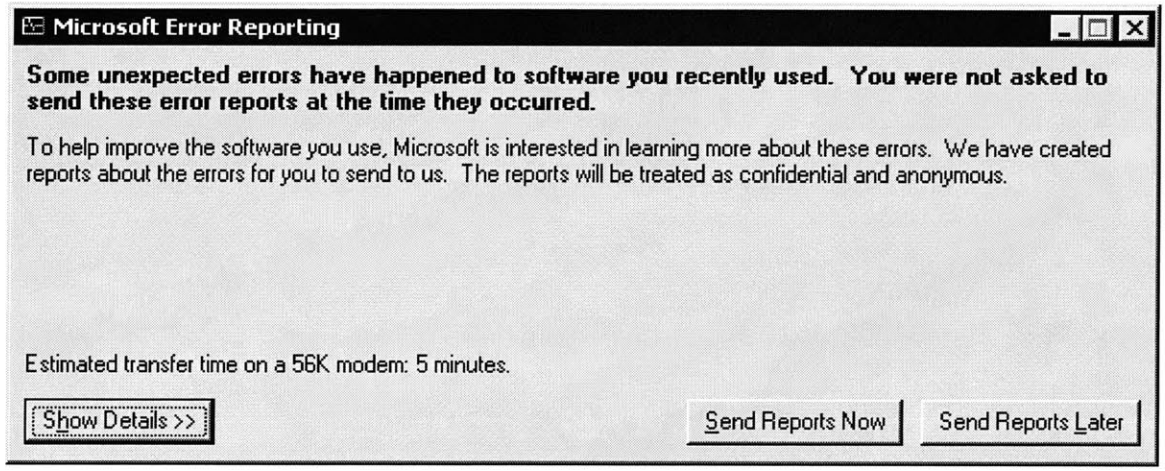


Figure 9: Initial Error Reporting Dialog Box

The detailed dialog-box provides information on when the errors occurred, which application encountered it, a brief description and the size of the error logs. The user has the option to view more details on this as well as view details of what data is being collected.

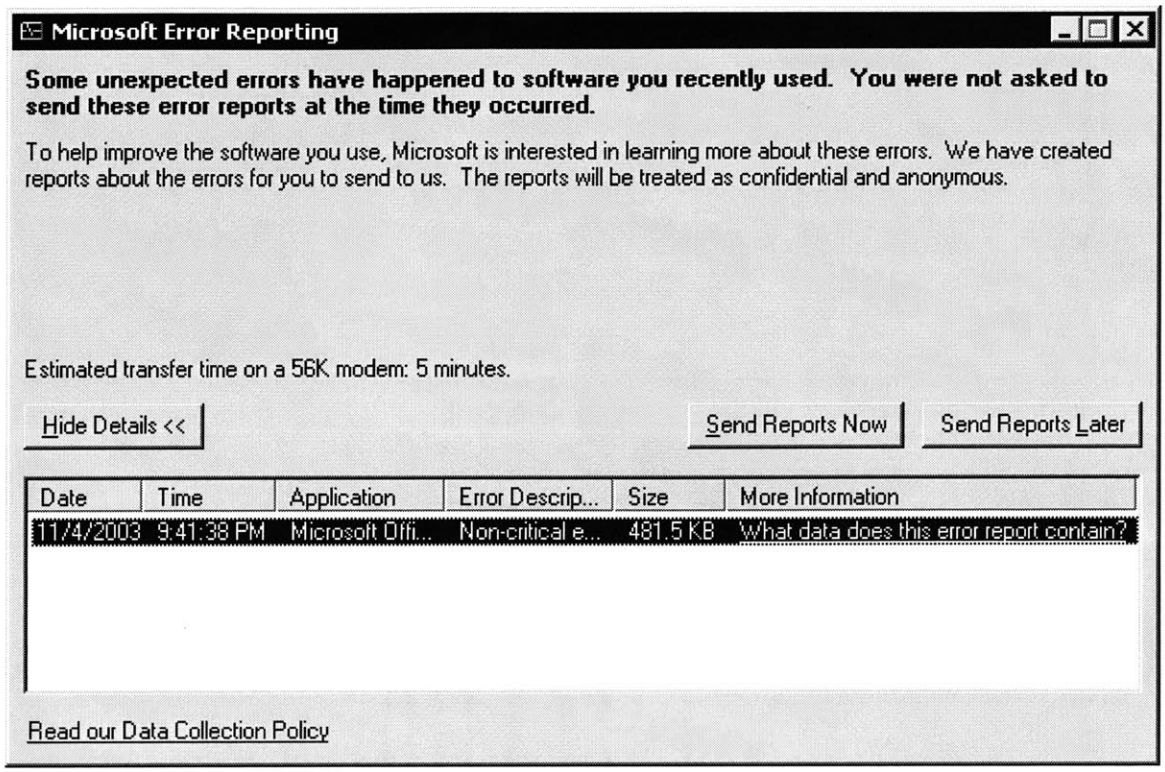


Figure 10: Detailed Error Reporting Dialog-Box

Clicking on the "Read our Data Collection Policy" link takes the user to the web page shown in the figure below. This page sheds more light as to what error data is being gathered and why it is being gathered.

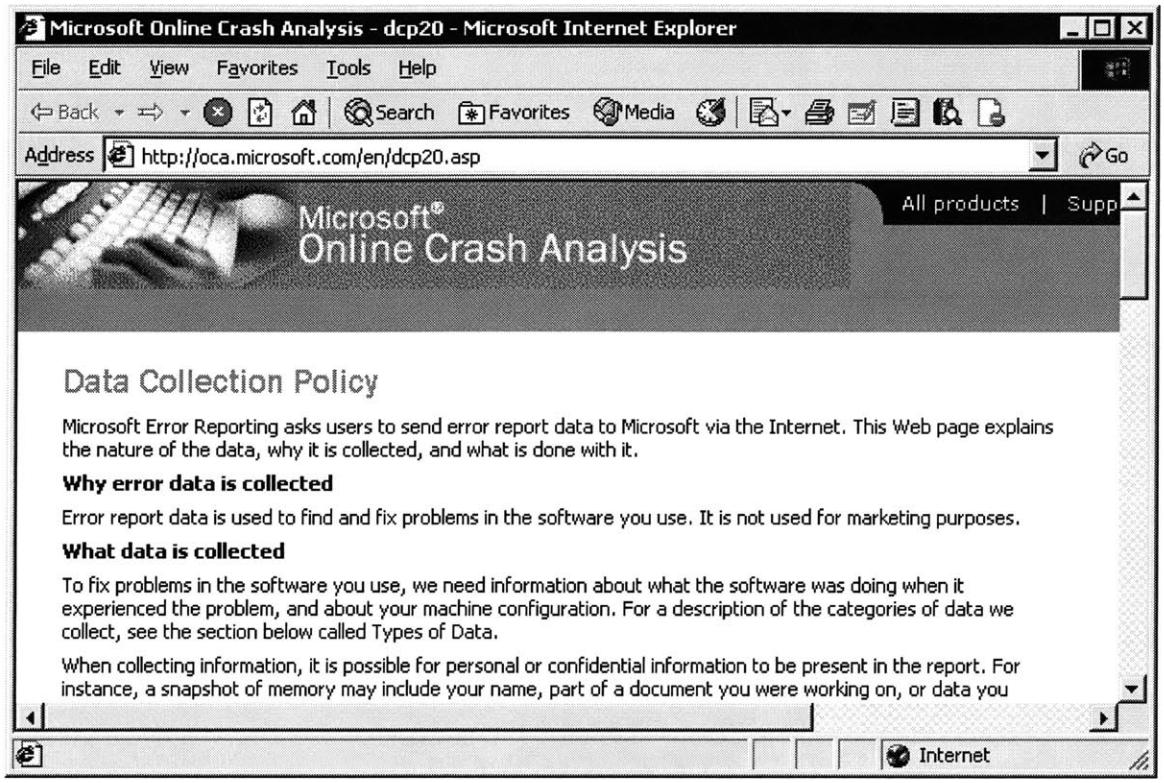


Figure 11: On-line Crash Analysis data collection policy

Clicking on the "What data does this error report contain?" link takes the user to the following Dialog-box. This box provides detailed explanation on the exact error, its signature, and reporting details.

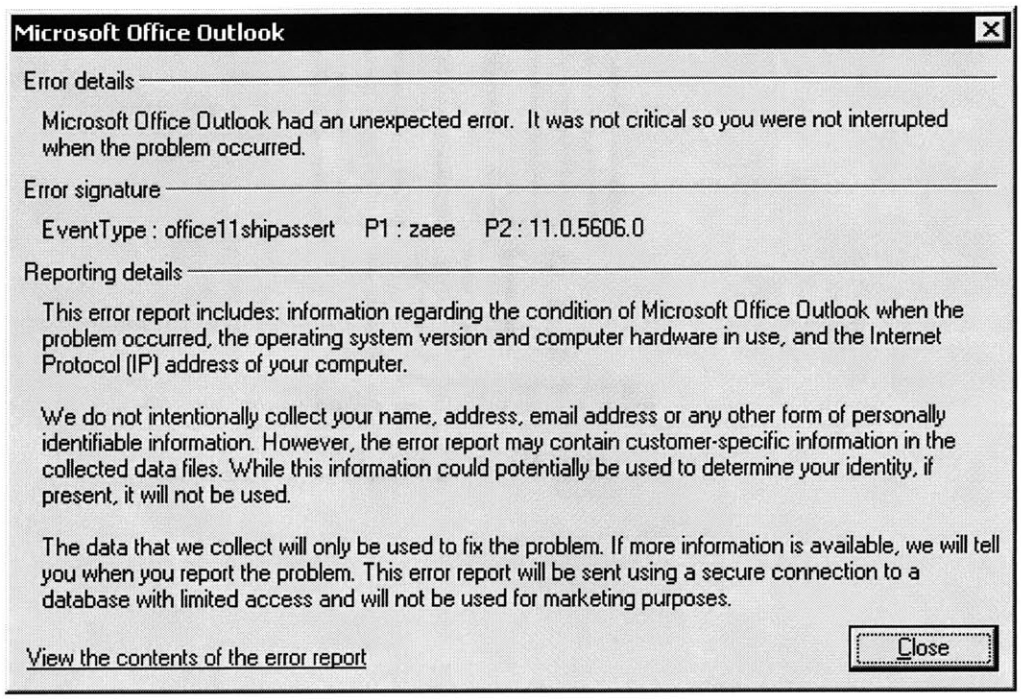


Figure 12: Details about the Error

Finally, clicking on the "View the contents of the error report" the user can view the actual files being uploaded to the Microsoft online crash analysis website as well.

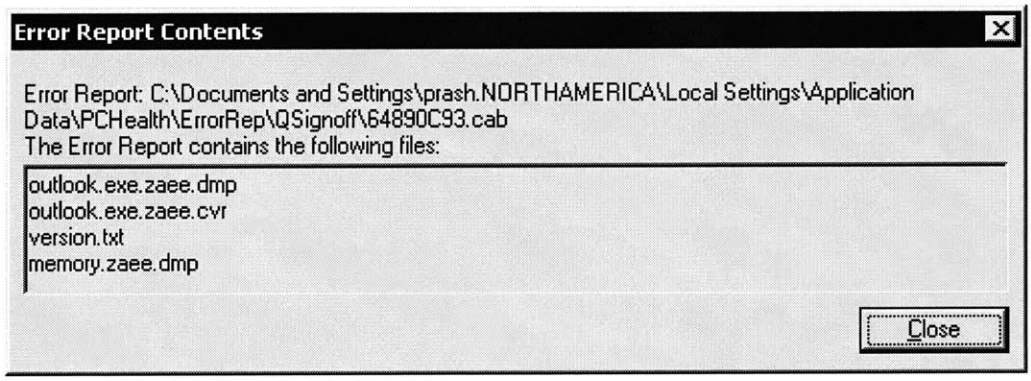


Figure 13: Contents of the error report

Improved error alerts from Office Watson

Office Watson is designed to provide alerts for errors in Office XP. It also provided reporting information back to Microsoft in order to improve Office. This edition of Office Watson provides additional local and Web Help information based on an error. This alert provides a more detailed error and informative error message with ways to resolve the error. This is an example of the self-help capability that is also integrated with the online Microsoft resources such Microsoft's online Knowledge Base (KB) or Online Help. This is evident in the URL embedded in the alert displayed in the figure below.

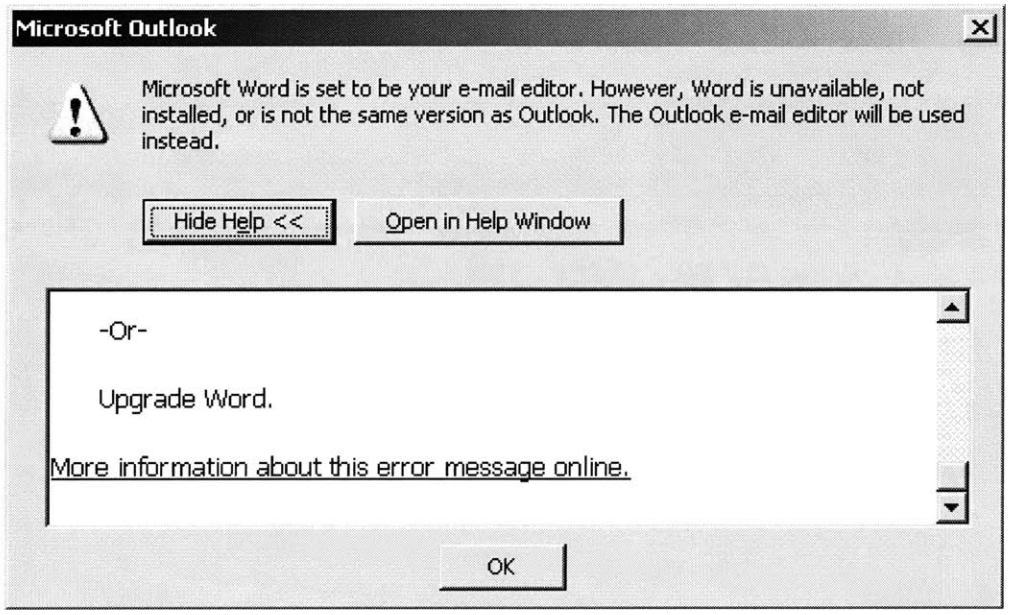


Figure 14: New Watson Alert

In order to generate meaningful information from the alert the information captured includes the name of the product, the Event ID, the Error Number, the language setting and Product version. It also possesses the capability to self-heal the system based on customer input and heuristic data. For example, it provides alerts for events that may cause future errors in Office programs and the user is provided with the possible decisions to tackle it. This figure below displays a known issue around an add-in and Microsoft Word 2003. The user is given the opportunity to disable the Add-in, continue loading the Add-in or view more information about the issue. The figure below shows an advantage of this self-help feature which is informing the user that the problem is a known issue and the steps required to resolve it.

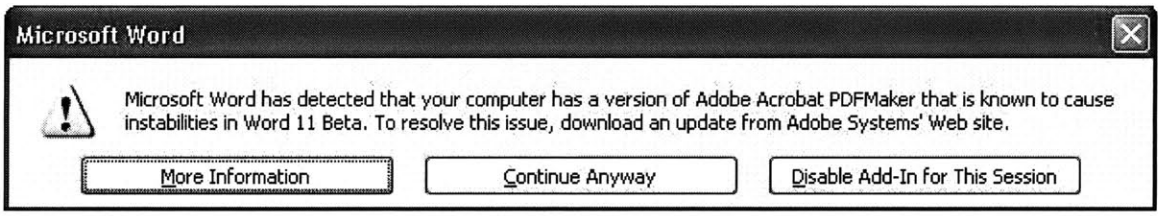


Figure 15: Known Issue with Add-in Watson Alert

Hang Reporting

Along the lines of the error reporting tool is the concept of a Hang Reporting Tool. In addition to determine a crash another a scenario that needs to be addresses is the ability to analyze applications that may not be responding in a timely manner. Such a capability could be expected to ship with the next version of Office and possible even the Windows operating system.

As the name suggests its objective is to determine which applications are responding in a timely manner and which are not responding in a timely manner. Once again as in the case of the error reporting tool, this tool is expected to run on the client machine.

Additionally, this capability is also expected to generate files such dumps etc. to help determine the cause of the hang, which can then be uploaded in order to analyze the data and determine root cause. An added possibility is that if the user wants to analyze the files they can save the files locally and perform their analysis.

3.2.3. Self-help

Office 2003 provides an improved help capability that has been extended to include an online self-help or assistance solution in addition to the help documentation installed with the product.

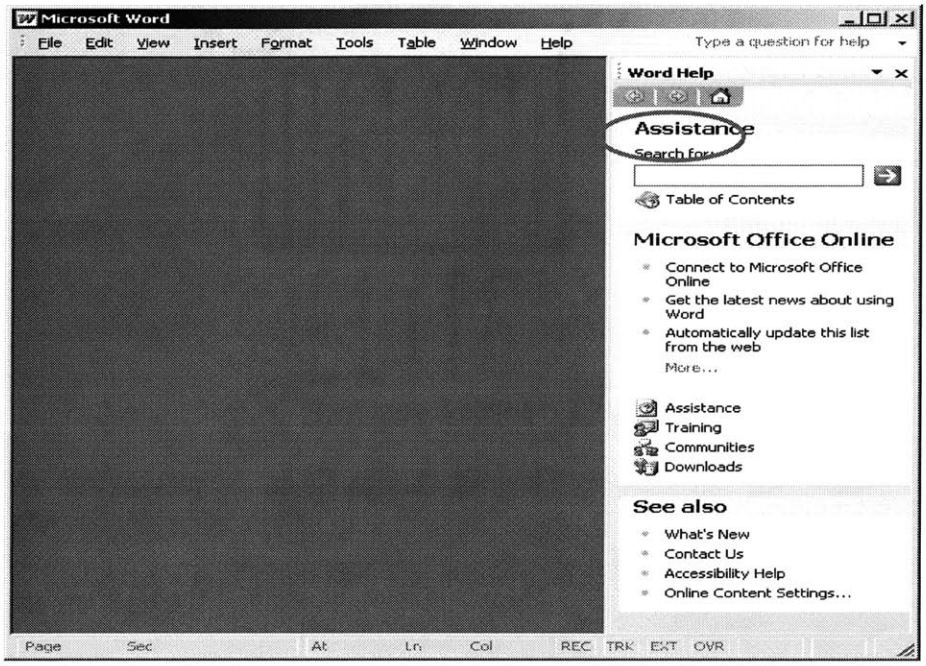


Figure 16: Office Self - Help and Assistance Capability

As the help indicates the Word help capability allows user to connect to Office On-Line help. An example is provided below with an error that occurred while reading a Word document. The self-help capability provides information as to how to recover from this problem. In this case the means to recover from the problem is by opening the Word document using the Text Recovery converter. It also provides exact steps on how to invoke the Text Recovery converter and what options to choose once it is started.

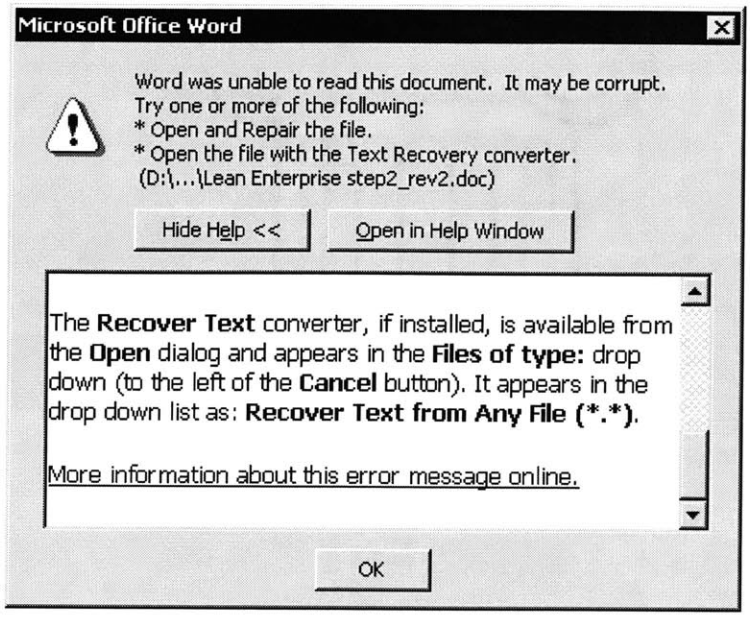


Figure 17: The self-help error message dialog box

The self-help dialog box also provides a link to the on-line Microsoft Support Center to retrieve more information on the error itself as shown in the figure below.

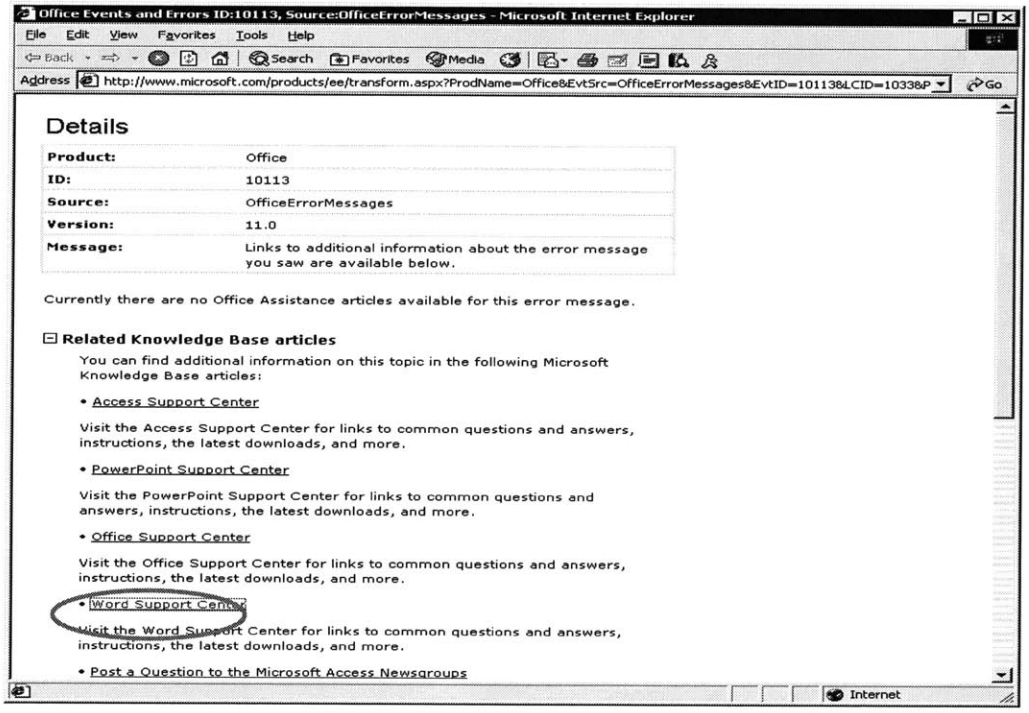


Figure 18: On-line self-help at the Microsoft Support Center

To get more details on the problem specifically related to one of the Office Suite of Products the user can select the appropriate Office Suite product. In the example provided it is related with Microsoft Word, hence selecting the Word related link takes the user to the Word support Center shown in the figure below:

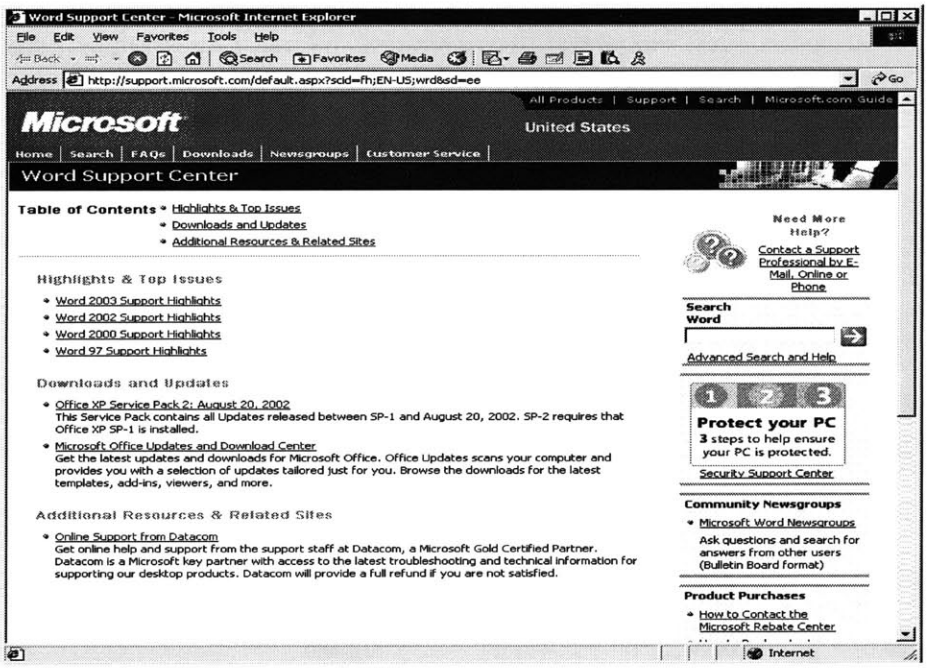


Figure 19: On-line self-help at the Microsoft Word Support Center

3.2.4. Proactive supportability Features:

Service Quality Monitor

The Service Quality Monitor (SQM) is a scalable and flexible means of gathering aggregate data about a user's activities, product usage, computer configuration and other important pieces of information that provide Microsoft with valuable information about Office. The SQM gathers information from Office Watson to record an accumulative file of events and errors. Periodically this information is uploaded to online Service Quality Monitoring servers. These servers are part of a new Customer Experience Improvement Program (CEIP), which as the name suggests are working towards improving the customer experience. Once a user has enrolled in CEIP, SQM captures data via the Cockpit Voice Recorder (CVR) on every event that initializes a Watson Alert. It does not gather Personally Identifiable Information. On capturing a compiling an error, SQM generates a detailed report, similar to the one shown below. In general the customer also has the option to enable SQM to upload this information to the SQM servers, in which case the Cockpit Voice Recorder file and a dump file uploaded. In case of fatal errors more detailed error information and additional information regarding the Operating System, Hardware, program and IP address related information are uploaded to the SQM servers.

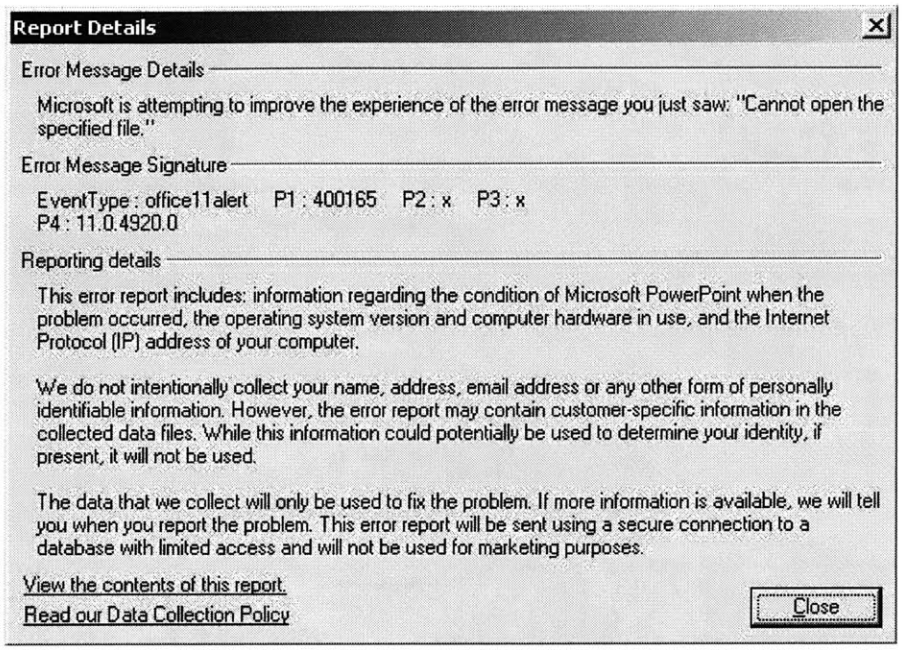


Figure 20: SQM Detail Error Report Alert

Online Customer Feedback

An additional feature provided by Office 2003 to customers, is the ability to provide more direct feedback to the Microsoft Office Development teams via the Microsoft Office Online. It offers Contacts, Feedback and Suggestions and Ratings pages. These pages empower users with the ability to let Microsoft know what they like, dislike and want to see in future editions of the Microsoft Office System.

3.3. Microsoft SQL Server

Microsoft SQL Server (pronounced as 'Sequel' Server) is a multi-user relational database management and analysis system for e-commerce, line-of-business, and data warehousing solutions developed by Microsoft. The current version of SQL server is 2000 and the next version is codenamed 'Yukon'.

3.3.1. Why is SQL Server now laying so much emphasis on supportability?

In the last few years a careful analysis of the customer cases revealed that the number of days required to find a solution, also called "Days to Solution" (DTS) to their problem was increasing. Additionally, case analysis also showed that the longer time it takes to solve a problem the less satisfied a customer tends to be with the solution. Finally, longer the time it takes to solve the problem, in other words longer the downtime of the customer's server, the higher the total cost of ownership for both the customer the SQL Server Support team. Hence, it was determined that in order to improve Customer Satisfaction and reduce the Days To Solution, there was a need to improve the supportability attribute of SQL Server. Additionally, it was determined that both the product support and development need to partner the responsibility of providing SQL Server supportability since the former was acutely aware of the 'What?' and 'Why?' of supportability while the latter had the expertise on 'How?' to incorporate it into the product.

3.3.2. How were supportability Features Prioritized

In order to prioritize supportability features extensive research was conducted on customer use cases, the amount of time and labor spent trying to resolve the cases. This resulted in identifying the nine major components or buckets of SQL Server that customers were contacting Product Support for namely, Storage Engine, Setup, Query Processor, Linked Services, Replication Services, Tools, Online Analytical Processes, and Transformation services. The remaining issues were bucketed into the miscellaneous category.

Next for each major category or component the number of issues **(I)** and the resources **(R)** expended to resolve respective issues was determined. All these factors were categorized based on each of the nine the SQL Server categories involved. Finally, based on the factors these components were prioritized to devise appropriate supportability features.

The prioritization shown in the table below is based on a variable called the **Supportability Factor (SF)** calculated as a weighted measure of the number of Issues (I) received, and the amount of resources (R) expended on the trying to resolve the issues. The data in the table below was for the year 2003 (January – October). Hence, for the nth component of SQL Server its Supportability Factor (SF) was calculated as:

$\mathbf{SF_n = f\,(\sum I_n, \sum R_n),}$ for the **nth** component of the product.

Rank	SQL Server Component	SF
1	Storage Engine	76.32
2	Setup	45.71
3	Linked Services	44.61
4	Query Processor	44.39
5	Miscellaneous	33.41
6	Replication Services	29.15
7	Tools	22.42
8	Online Analytical Processing	17.51
9	Transformation Services	12.12

Table 3: SQL Server supportability Issues based on SF for 2003 (Jan-Oct).

In addition, to the buckets that comprise internal SQL Server components, a similar approach was used to prioritize external components that customer use to access SQL Server called as Data Access Technologies. Data Access Technologies were bucketed into Extended Markup Language (XML) and Microsoft Data Access Components (MDAC). As before the prioritization shown in the table below was based the **Supportability Factor (SF)** calculated as a weighted measure of the number of Issues (I) received, and the amount of resources (R) expended on the trying to resolve the issues. Hence, for the nth component of Data Access Technologies its Supportability Factor (SF) was calculated as:

$\mathbf{SF_n = f\,(\sum I_n, \sum R_n)}$, for the **nth** component of the product.

Rank	Data Access Components	SF
1	MDAC	295.32
2	XML	63.87

Table 4: Data access supportability issues based on SF for 2003 (Jan –Oct).

Additionally, to harness the partnership between Product Development and Support, Product Support was provided a special channel to directly convey such supportability related data to development, by means of a supportability database of bugs. These bugs contain the problem scenario, the list of issues that encountered this problem, and the recommended steps that need to be taken to improve the timeliness in solving the customer issue. This provided a process for a feedback loop to Design and Development for non-bug related issues which nevertheless help in improving the product in terms of supportability. Additionally, in order to measure progress, each bug has a scorecard associated with it. Each component of SQL server has bugs filed against it and each bug is prioritized in order to ensure that the most critical ones get the most attention.

3.3.3. Supportability Objective

Based on the 'What?' and 'Why?' covered before an objective was framed which comprised improving Customer Satisfaction and reducing the "Days to Solution" for incoming customer cases. To achieve this objective it was devised that first the customer needs to be empowered or better equipped to solve problems themselves and second Product Support needs to be empowered so that when a customer contacts Product Support they support personal should be able to resolve the issue in a timely, efficient and effective manor. The first aspect was called Serviceability, which is the ability for the customer to deploy and maintain the product with the world's best, out-of-the-box solutions and the second aspect was supportability, which deals with the information and tools necessary to support the product in a timely, efficient and effective manor.

3.3.4. Supportability as a whole

The first step in this initiative to kick-start the serviceability component of SQL Server that comprises all the help the product can provide to the customer so that they can be empowered to solve a problem when it happens without having to call into Microsoft Product Support. This primarily comprises improving the error messages and improving the self-help capability. Improving error messages entails providing more informative and understandable error messages instead of the incomprehensible "Errors Occurred" or "General Network Error". The self-help consists of the capability of the product to help the customer use the information provided by SQL Server Error Report to find out the cause and how to fix it. An example of an error message is:

Error 1223: "Process ID %d:%d cannot acquire lock "%s" on resource %s because a potential deadlock exists on Scheduler %d for the resource. Process ID %d:%d holds a lock "%h" on this resource."

This error is extremely helping in deadlock scenarios, where the deadlock has occurred, who are the exact culprits, determined by their Process Id's and thread ID; involved in the deadlock, as well as the type of lock requested by the culprits. Such data is extremely helpful to SQL Server Database Administrators who can quickly determine that cause and take measures to ensure the culprits do not run at the same time, without even having to call into Product Support, which had been the only alternative before.

The next step deals with effectively solving customer problems in a timely manner once they call into Product support. The key concern of SQL Server support is issues that require extensive use of the debugger. This primarily stems from the fact that such issues tend to take an inordinate amount of time to solve. Hence of the key initiative is to try to solve the case without using the debugger as much as possible. This has been achieved with co-operation from the development where extensive logging or divulging the metadata has been implemented in key places in the code where more information is needed to successfully solve the problem. This has resulted in the enhanced SQL server process health monitoring. Health monitoring enhancements have taken place in the following areas: Blocking, Network problems, Input/Output (IO), Memory and CPU. When SQL Server detects health problems, a series of new error messages are logged in the SQL Server error log.

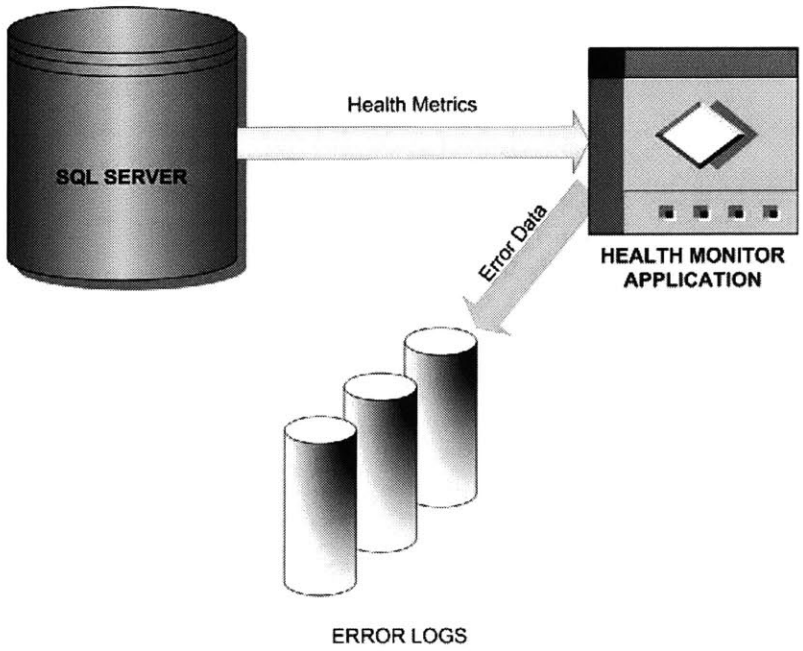


Figure 21: SQL Server Health Monitoring Architecture

For example SQL Server now provides proactive I-O auditing or logging data when SQL Server is actually reading and / or writing physical data in the form of pages to the hard disk. It is imperative to have some amount of transparency in this operation since it is key to many issues dealing with database corruption, database hang etc. The current implementation determines whether the data on the page is correct by way of a checksum and if any problems are encountered it now logs all this data into an error log.

Here is a sample scenario where this helped solve a customer issue: A customer was receiving errors that the SQL Server was hanging and unable to completed processing of incoming database requests. Looked at logs it was determined that apparently SQL Server was in fact

waiting for the operating system to return confirmation from its Hard Disk Driver that the data had been read successfully or not. On checking with the Operating System team it was determined that the problem was with the driver and an updated driver was sent to the customer to fix his problem. Such improvements have led to improving not only supportability for new product releases but also older legacy products. This in turn helps improve the stability of the server product, which is required in products that guarantee hard real-time guarantees to customers on metrics such as total maintenance downtime in a year, which in some cases stands at 2 hours.

Self-Tuning and Self-Administering Databases [11]

Database management systems provide functionality that is central to developing business applications. Therefore, database management systems are increasingly being used as an important component in applications. Yet, the problem of tuning database management systems for achieving required performance is significant, and results in high total cost of ownership (TCO). The goal of our research in the AutoAdmin project is to make database systems self-tuning and self-administering. We achieve this by enabling databases to track the usage of their systems and to gracefully adapt to application requirements. Thus, instead of applications having to track and tune databases, databases actively auto-tunes itself to be responsive to application needs. Our research has led to novel self-tuning components being included in Microsoft SQL Server.

3.3.5. Organizational and Cultural Alignment

In order to emphasize the importance of supportability, the expertise and experience of the support group was sought to pin-point supportability features on various components early in the development cycle of future products. This partnership entailed key, highly-regarded and experienced support professionals dedicating a portion of their time to solely reviewing the source for a component of the server and providing feedback to the development organization as to what would help improve supportability of the product and why.

Additionally, to build up on the supportability initiative the SQL Server development group implemented organizational changes. They tried a centralized as well as a de-centralized approach. The centralized approach entailed having one individual called the supportability Program Manager owned the supportability feature set for the entire SQL Server Product. This supportability feature set comprised owning and driving the maintenance of legacy code as well as the new supportability features being implemented in upcoming service pack and new product releases. In addition, to the vast ownership boundary was the daunting task of managing this across a very complex, componentized product where each component of SQL server was essentially owned by a separate development group within SQL Server. Once the scope and enormity of the task was determined, a new decentralized approach was practiced where the Program Managers leading the development effort on a particular component would own the supportability feature-set for that component. However, this is not without its drawbacks, primarily that the level of supportability may not be consistently implemented across all components of the SQL Server product.

One of the key concerns is sustaining the drive to improve supportability. This stems from the fact that interest in it can potentially wane since such feature do not key visibility in terms of progress unless customers run into them. This generally happens only when the product has been released to the customer. Hence joint ownership between development and support on supportability features is important to ensure such features ship a constant.

3.3.6. Effectiveness of some Initial supportability Features

Data dealing answering how effective some of the supportability changes have been in SQL Server 2000 have been favorable. Data in terms of the amount of labor expended was gathered with regards to resolving issues dealing with two particular SQL Server components in which supportability features were introduced. The following graphs illustrate by how much the labor was reduced for solving issues dealing with two SQL Server Components. Both the graphs clearly indicate that after the second supportability related feature was introduced there was a gradual reduction in the amount of labor expended in resolving issues dealing with the same two SQL Server components.

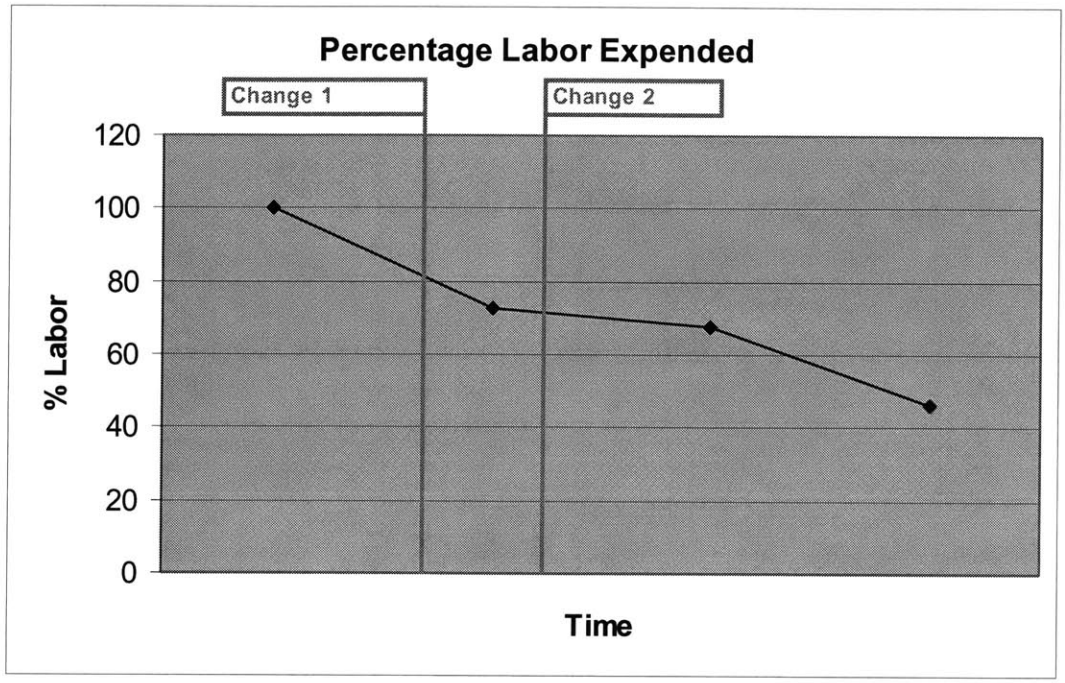


Figure 22: **Supportability effectiveness for the first Component**

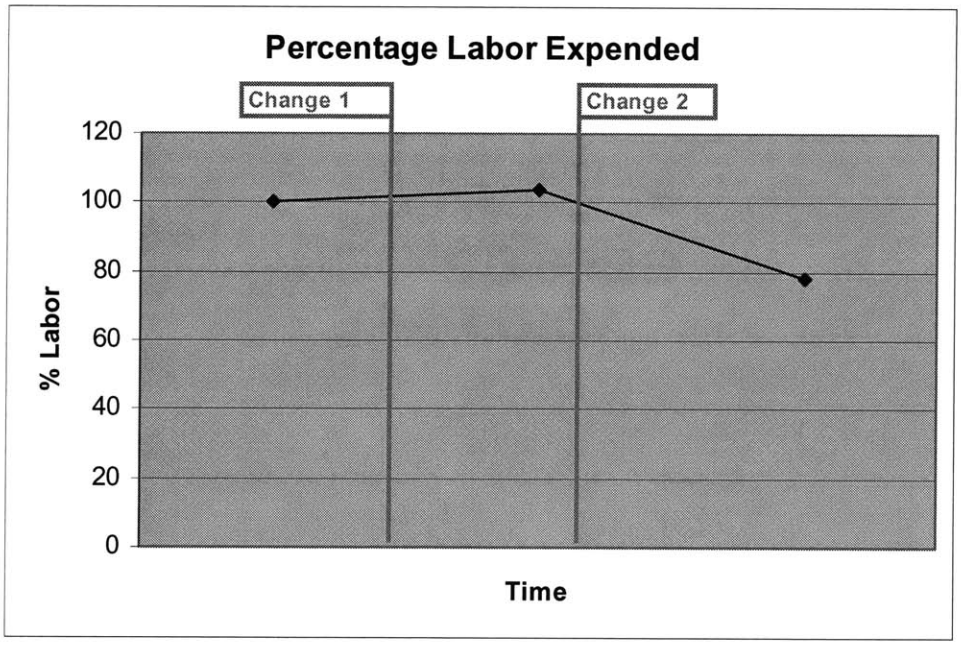


Figure 23: **Supportability effectiveness for the second Component**

3.4. Microsoft IIS Server

Microsoft IIS stands for Microsoft Internet Information Server. It is Microsoft's version of a Web Server. It provides a Web application infrastructure for customers host their web applications and web pages. The current version of the IIS server is 6.0.

3.4.1. How were supportability issues prioritized?

This involved a lot of groundwork that entailed working with Microsoft Product Support in analyzing what problems currently customers are facing as well as what issues Product Support had difficulty supporting with regards to the previous versions IIS 5.0 and the newly released version IIS 6.0. Each technological area was ranked on the basis of how many incidents were received, and the amount of labor related resources expended on it. The next 2 tables list the major issues encountered with the IIS 5.0 and IIS 6.0 Web Servers. The prioritization is based on a variable called the **Supportability Factor (SF)** calculated as a weighted measure of the number of Issues (I) received, and the amount of resources (R) expended on the trying to resolve the issues for a particular component of IIS. The higher the SF the higher is the priority to improve that particular components supportability. Hence, for the nth component of a product its Supportability Factor (SF) can be calculated as:

$\mathbf{SF_n = f(\Sigma I_n, \Sigma R_n)}$, for the **nth** component of the product.

Rank	Component	SF
1	Third Party Related Issues	8.07
2	Transaction Services	5.13
3	Security	4.22

Table 5: IIS 5.0 supportability Issues based on SF.

Rank	Component	SF
1	Security	26.63
2	Windows 2003 Server	1.67
3	Transaction Services	0.90

Table 6: IIS 6.0 supportability Issues based on SF.

3.4.2. Supportability Objective

The supportability initiative in the IIS team essentially boiled down to improving overall the Quality of Service offered to customers. Simply put, supportability for IIS comprised all the key areas that make IIS supportable by customers as well as by Product Support. This implied that

initially the customers should be able to use self-help and self-diagnosis functionality before calling into Microsoft Product support Services.

Additionally, based on a detailed study of bug and customer issues it was clear that the supportability solution needed to be mindful of specific issues such as IIS server hangs, memory leaks, memory corruption, slow performance and crashes were the major issues that a lot of labor and time were spent on. Hence, improving overall customer experience would require addressing such scenarios that adversely affected Quality of Service. In order to do so the followings needs had to be kept in mind:

- Resolve issues in a timely manner with the increasing complexity of customer scenarios,
- Make troubleshooting data collection easier by being mindful of the debugging skill level of the frontline support personal and /or the customer. In other words they may lack hard core debugging skill, limited troubleshooting skills.
- Collect useful data as early as possible in the issue. Studying the cases it was found that a lot of cycles were spent going back and forth between the customer and the support professional to get the appropriate troubleshooting data that eventually solved the customer's issue, which essentially is the Muda of Rework [40].
- Ease for the customer to self-solve the issue using tools and proactive data collection. There was a need to share ownership in the joint development of this supportability initiative between the IIS Development and Support teams.

A non-design related issued that surfaced was the training of Support personals on customer management skills by managing customer expectations and coordinating conflicts with early management involvement.

In addition to improving the Quality of Service, another major driver of supportability is Total Cost of Ownership for Microsoft and Microsoft's customers. The Total Cost of Ownership started increasing since customers started running to more complicated scenarios and Product Support started running out of resources i.e. time and personnel; to solve the problems encountered in the complicated issues. Hence the concept of empowering customers to self-help and self-diagnose the issues upfront via the design of the product; even before calling into Product Support took shape.

3.4.3. How is IIS currently improving "After-the-fact" supportability?

The first step is the Internet Information Services (IIS) debug tool suite is targeted to troubleshoot IIS crash as well as hang customer scenarios. It comprises:

- **IIS Crash/Hang Agent** tracks all incoming and outgoing IIS requests. It also possesses the capability to log requests that are pending during an IIS process crash. This tool also detects which requests that are hung and allows the customer to take action when such a situation is detected.
- **IIS Dump** is a command line tool for gathering information on Hangs. This tool aids root cause analysis by proactively extricating information on different IIS

related components in a manageable, convenient and readable set of formatted outputs.

A favorable outcome observed was a standardized troubleshooting support process resulting in better customer experiences. It also afforded gathering better statistical data that can be fed back to R&D and support for further improvement in the customer experience.

3.4.4. How does IIS plan to improve supportability as a whole?

The next version of the IIS server goes beyond simply facilitating after-the-fact troubleshooting. It contains a proactive Server Health Monitoring capability, after-the-fact Error Reporting and Recovery, and after-the-fact Troubleshooting and Debugging components.

The monitoring component works on answering questions such as 'how can customers figure out that their servers are healthy or not?' and 'what can IIS provide that will assist with figuring out the health of the server?' The current solution is four fold. The first is a monitoring component – Microsoft Operations Management Utility. This requires creating rules, which help the MOM utility determine when an IIS server is in good or bad health. When bad health is detected then the MOM utility raises an alert to inform the customer on the bad health of the customer. The second is a gathering appropriate performance counters. The third is logging data that can be used for many different things, including usage analysis, intrusion detection, etc. and finally a component that informs the customer what request the server is processing at the present time.

The Error Reporting and Recovery capability stems from making error messages more meaningful and actionable. This includes errors such as "Unspecified Error" to errors that indicate what the exact error is and why it was thrown. Additionally, it also possesses the capability to automatically recover from such failures.

The Troubleshooting component encompasses questions such as 'how can our customers and Product Support Professional figure out what went wrong, when a problem occurs?' and 'how does a customer correct the problem once they have a diagnosis to the problem?' Based on the initial groundwork performed analyzing customer issues two major troubleshooting categories were determined. The first type of issue was caused by incoming requests from clients that resulted in erroneous or unexpected outgoing responses from the server. The current solution to trace or log a request when it is received by the server all the way to when and where the "unexpected" behavior is occurs. The second set of issues was caused by those that do not deal with requests i.e. they are non-request-based issues. The current solution is to instrument the server to log or trace causal diagnostic information at deterministic points of failure in the code which have traditionally caused customers and Product Support difficulty troubleshooting, for example, when the server or its worker process fails to start up or shut down in time, or the server's administration console refused to started etc. This traces or logs provide the additional information for analysis issues such as server crashes.

Another troubleshooting aspect revolves around answering the question 'how can Product Support solve the customer's problem faster'. This approach includes training and educating

Product Support on the various Troubleshooting technique and which technique to use for which problem. Another major concern raised by Support Professionals was that they had to frequently refer to the IIS source code in order to find out the answers to the customer's problem. This required in-depth knowledge of the source code, which is extremely large in SLOC and complex. Hence, the new training involves educating PSS into new troubleshooting techniques that virtually eliminate the need to refer to source code in order to solve the customer's problem, reducing the time to solve such customer problems.

The Debugging component is concerned with debugging live applications easier. Based on the initial analysis it was found that a lot of times customers needed debugging when their servers were running in their production environments. This would be difficult since debugging has a high cost in terms of performance which may not be a favorable alternative in production scenarios. Currently the IIS Debug tool is the solution in this regard and a complete solution still is being conceived.

3.4.5. Organizational alignment

Originally the design and development team that developed the software moved on to develop the next version of the IIS Server once it finished with the current version. A completely separate Sustained Engineering would handle the maintenance of the 'shipped' versions of the IIS server. Hence any maintenance related issues were subjected to a learning curve since the people who wrote the code were no longer involved in its maintenance. The Development team is moving

away from such a 'Skunkworks' mindset. This mindset has changed in that the Sustained Engineering Team is now more involved with the Development team and vice versa. Finally, long term goals point towards considering merging both into one team.

Additionally the IIS team now has a Program Manager on the Development and Design Team that is dedicated towards supportability Features of the product. All the end-to-end supportability capabilities discussed in this section including Monitoring, Error Reporting, Troubleshooting and Debugging fall under the aegis of this Program Manager. This is along the lines of the Multi-Discipline Teams where each member of the team assumed ownership of an 'ility'.

3.5. Non-Product Aligned initiatives improving supportability

3.5.1. Microsoft Operations Manager (MOM)

Microsoft provides a solution to customers for proactively monitoring called *Microsoft® Operations Manager* or *MOM*, which is an enterprise operations management tool that helps IT personnel to manage the availability and performance of their systems in a cost effective manner. It is primarily aimed at NOC operators, service owners, IT managers and CIO. However, it can also be used by support personnel. Actually only about 1% of the problems faced are real bugs that need a QFE, the rest are operational problems for which a workaround, a best practice or a corrective configuration that already exists. MOM addresses the 99% of the problems that usually the customer fends for on their own. MOM manages 26 different apps and services from Microsoft including Microsoft SQL Server, IIS Server, Exchange Server etc. Third party Management Packs are also available to manage 20 other applications and hardware nodes.

The primary objective of MOM is to monitor a server's health. Good health is determined by a number of processing rules that are broadly categorized into Event, Performance, Alert Processing. This information base of all the rules is called the Knowledge Base.

- **Event rules** comprise Collection, Prefilter, Postfilter, DB filter, missing event, consolidation and event processing rules. These can vary from server to server. For example; an IIS Server event rule might require looking for a HTTP 404 error

in the IIS Logs, while a SQL Server event rule might require searching the Windows Event Logs for an AD Replication Error. These processing rules are authored by the Development teams, customers and product management for the respective server applications.

- **Performance rules** comprise thresholding rules and sampling rules, both of which can preprocess the collected data stream. For example; IIS Server might setup a rule to determine when the response times for incoming requests exceeds a healthy threshold. Essentially they determine what data needs to be sampled and what are thresholds that need to be met to decide whether any rules are getting violated or not. MOM periodically samples instrumentation data that help measure that threshold of each rule. Instrumentation data is gathered from the server application as well as the machine on which it resides. MOM taps into existing instrumentation data generation capabilities of the Windows Operating System for such information. This includes tapping into three sources which are Performance Monitor, Windows Management Instrumentation (WMI) logs and script generated logs. If the server is unable to glean performance monitor or WMI related metrics then MOM possesses the capability to implement its own timing instrumentation using scripts to detect when a server is successfully running and when it is failing. The diagram below illustrates a MOM rule with a threshold rule of 50 when taken over 2 samples:

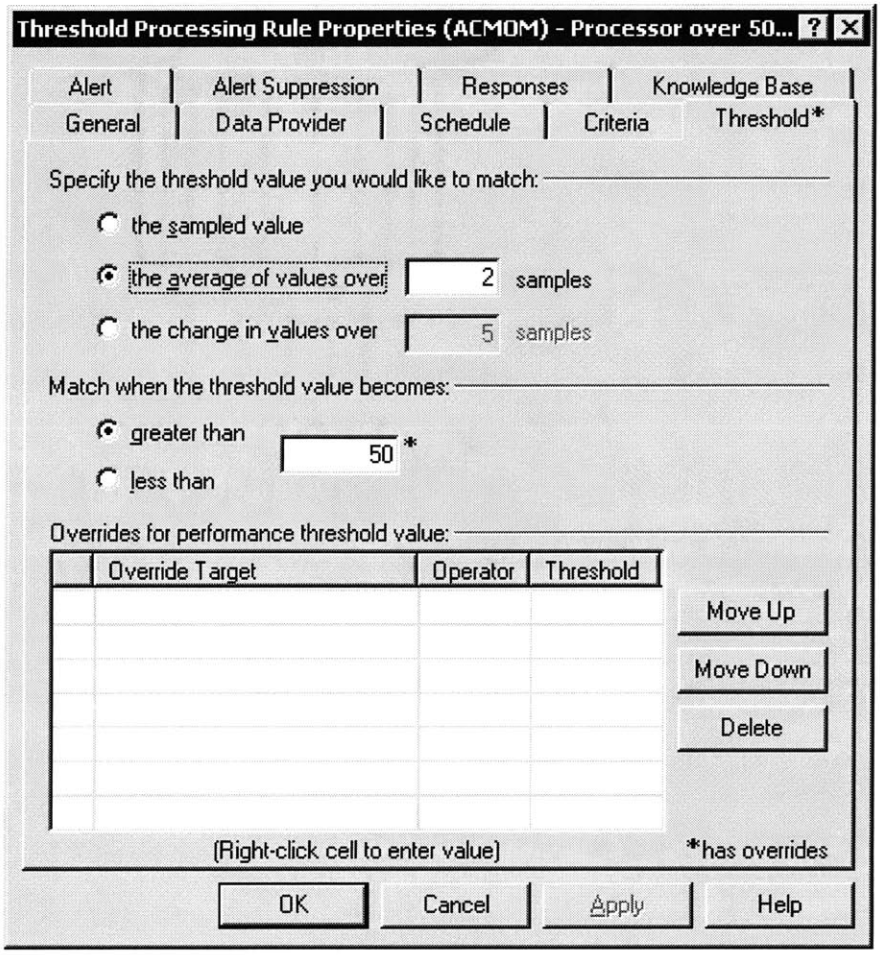


Table 7: Example of a MOM threshold rule

- Finally the last set of rules is called **alert rules**, which essentially act on alerts of a certain type. The event and performance rules generate alerts and responses. All event or performance rules need not require an associated alert rule. An alert rule is used to collect all alerts of a certain type and take an action. The different types of action could include, alerting the customer by way of a message box or emailing or paging the customer, or logging the data associated with the current

event, or running a custom script that automate steps that need to be taken to alleviate the problem. These scripts could include automating repeatable scenarios that the customer encounters.

Based on the rules discussed above a very large number of data points, events, samples and probes are generated. MOM takes all that data and does a 10,000 to 1 compression to only alert the customer or the network or database administrator of a real issue that needs action on their part. The important point is the ability to automate the response when a problem is inferred. MOM affords this capability to its user. The total number of rules MOM allows a server to target stands at around 20,000. In case of a large environment of around 7000 servers it was found that in excess of 200,000,000 events can be received per day, which can result in 1500 alerts using MOM noise filtering capability, which are further reduced by end user interventions to for example, 200 genuine problems issues at Microsoft per day.

The MOM architecture includes various components that are deployed in configuration groups. The basic components in a configuration group are:

- MOM Database, which is a Microsoft SQL Server database that stores configuration and monitoring data,
- MOM Administrator console, which provides a MOM user interface
- MOM DCAM, which is the central MOM server on which the MOM Administrator console is installed. It comprises three components, a Data access server, Consolidator and Agent Manager.
- Agent applications that run on each monitored computer

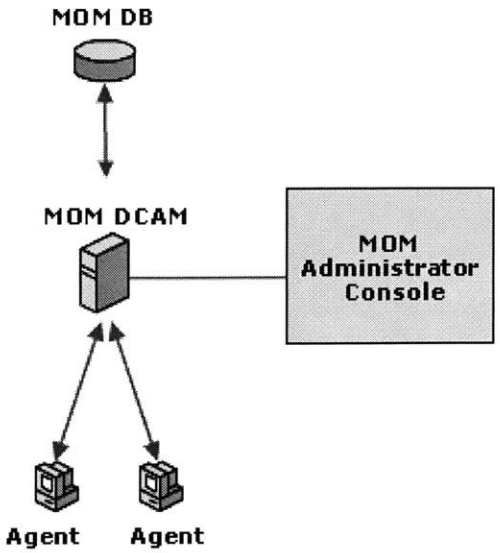


Figure 24: MOM Components

The entire sequence of MOM monitoring can be primarily broken down into three cycles. The first cycle is to discover machines and auto-install agents on them. The second cycle is to discover applications on each machine system and send down the appropriate rules. The third cycle is to collect data, filter it and act on it based on a rule. This third cycle is illustrated in the figure below:

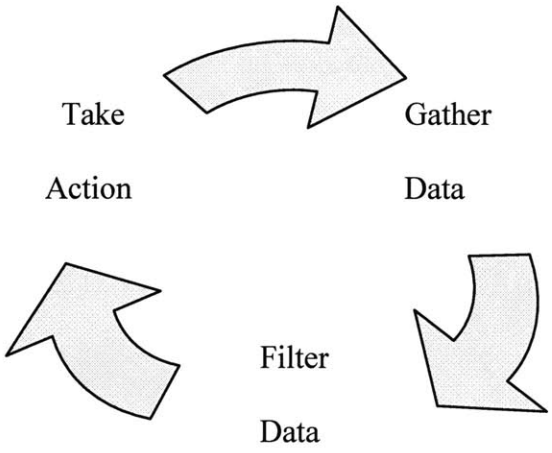


Figure 25: Third Cycle of MOM Monitoring

Some scenarios may comprise two additional cycles. The first of which could include MOM alerting the database administrator via an email, or page, or the MOM console and they call a MOM task to diagnose and correct the situation. The next such cycle could occur when periodically MOM uploads aggregate reports back to Microsoft, the various development teams study issues that affect QOS and TCO intensive and they correct it in MOM knowledge base and the product, which could then be downloaded over the web.

3.5.2. Bayesian Network Editor and Tool Kit (MSBNx) [12]

MSBNx is a component-based Windows application for creating, assessing, and evaluating Bayesian Networks, created at Microsoft Research. A common use of Bayesian belief network models is to diagnosis system failures in a probabilistic framework. This has several advantages over conventional rule-based or decision-tree methods, since Bayesian networks support uncertain evidence in a theoretically correct fashion. In addition, prior distributions in Bayesian

networks can be built that model logical functions such as AND, OR and NOT using what are known as deterministic nodes; that is, nodes whose distributions contain only zeroes and ones. Such nodes, therefore, act as logic gates. The application is helpful to designers and developers in making for informed decisions and its components run on Windows 98, Windows 2000, and Windows XP.

MSBNx is a tool for doing this kind of cost-benefit reasoning for diagnosis and troubleshooting. If you provide MSBNx cost information, it performs a cost-benefit analysis. If no cost information is available, MSBNx makes recommendations based on the Value of Information. It supports Discrete Sparse and Causally Independent Probability Distributions. Additionally, it conducts Probability Assessment using Standard, Causally Independent and Asymmetric Assessment methodologies. Here is an example of a question it tries to answer:

Q. When your car doesn't start, what should you do?
- Check if the lights work, or
- Replace the fuel pump.

Answer a) seems wiser because checking your lights tells you much about your car's battery, a likely cause of your problem. But that's not the whole reason to check your lights first. Just as important, you check your lights first because such a check is easy and inexpensive. Here is how the example would be graphically depicted in the Bayesian Belief Network:

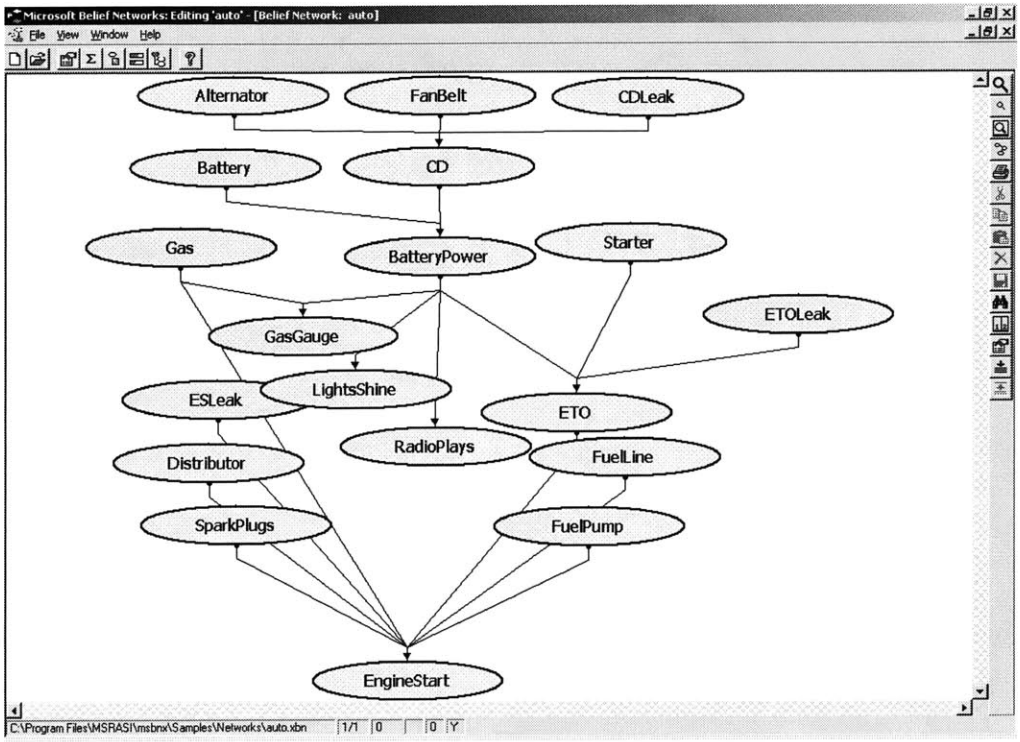


Figure 26: Bayesian Belief Network example for starting an automobile engine.

Based on the Belief Network this tool would evaluate the question as to what needs to be done when the car does not start. In the simplistic case where the only two alternatives are being compared i.e. to check the lights or the fuel pumps, the evaluation states that if the engine does not start then the probability that the Lights do not work is 0.4240, while that of the Fuel Pump being bad is 0.1943. Hence, the probability that the Lights do not work is higher, one should check the Lights before checking the Fuel Pump. This example is depicted in the figure below.

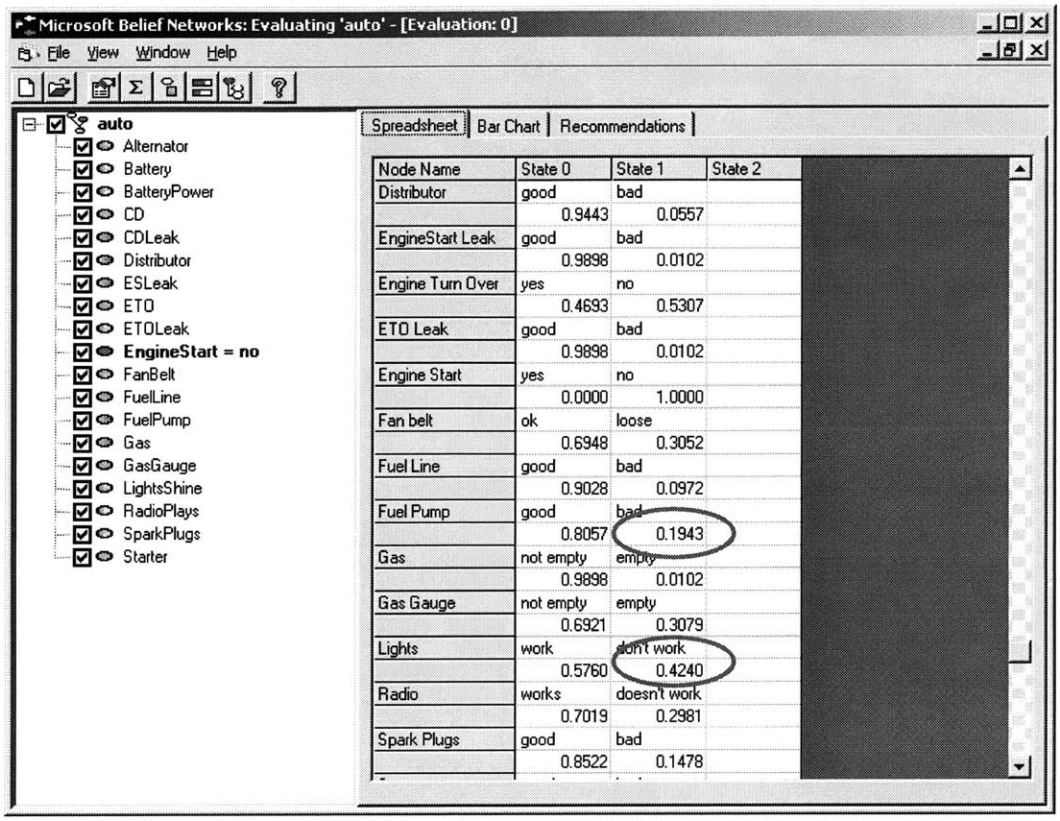


Figure 27: Evaluating using Microsoft Belief Networks

3.5.3. Troubleshooting using checkpoints [13]

Application failures characterized by the phrases, "it worked yesterday, but it doesn't work today" and "it worked on that machine, but it doesn't work on this machine" are a major source of computer user frustration and a major component in the total cost of ownership. The typical symptom-based troubleshooting approach relies too much on creative thinking and may lead users or support technicians in directions far from the actual root cause. Wang et al [13]propose a

state-based troubleshooting approach for configuration failures that aims at making the diagnostic process as mechanical as possible.

Their state-based troubleshooting approach entails two phases, the narrow-down phase and the solution-query phase. In the narrow-down phase the authors use checkpoint comparison and application tracing to determine which pieces of persistent state have changed and are affecting current application execution; ongoing self-monitoring of persistent-state changes by the machine is used to help eliminate false positives. In the solution-query phase, state-to-task mapping and searches of online databases are used to translate low-level state information into high-level user interfaces and articles.

Wang et al [13]also describe the design and implementation of a troubleshooter that uses their state-based approach. The data for each persistent-state is obtained from that generated by the System Restore utility that ships with Windows XP. This data comprises information related to the Windows registry. In some cases this data can be extremely large and hence according to the authors, the major challenge is to narrow down the root cause from a large set of registry values. One of the methods practiced in narrowing down the registry values is finding unique registry values 'N_d' amongst the persistent states. In addition to the checkpoints it also allows the scenario to be run so it can trace which registry keys are being touched. An intersection between the two provides 'N_T'. To narrow down further the authors use the Inverse Document Frequency (IDF) technique to calculate a score for each registry value. This technique is used to filter out any false positives from 'N_T' to 'N_I'. Finally, if after this there still are multiple registry values that remain then the tool applies an ordering scheme 'N_O' using which the user can make a more

informed search for solutions such as at the online Microsoft Product Support Services Knowledgebase. Here is a case study of for scenarios where this tool was effective in narrowing the cause:

Application Scenario	N_d	N_T	N_I	N_O
PowerPoint Slide Show	2,015	2	2	1
Instant Messenger Performance	2,329	21	9	1
Media Player "Open URL"	72,656	12	12	1
Word "Preparing to install"	1,989	4	1	1

Table 8: Summary of Case Studies

3.5.4. Detours [16]

Detours is a library to instrument arbitrary Win32 functions on x86 machines. Detours intercepts Win32 functions by re-writing target function images. Innovative systems research hinges on the ability to easily instrument and extend existing operating system and application functionality. With access to appropriate source code, it is often trivial to insert new instrumentation or extensions by rebuilding the OS or application. However, in today's world of commercial development, and binary-only releases systems researchers seldom have access to all relevant source code.

The Detours package also contains utilities to attach arbitrary DLLs and data segments (called payloads) to any Win32 binary. This is generally used for troubleshooting purposes to determine root cause of a problem by developers, testers and support personal in tracing existing source code that has generally been deployed to the customer site.

3.6. Concluding Observations

Depending on the functionality of the product and customer base the supportability needs may vary. However, the supportability objective posses a common thread of improving the Quality of Service for the customer despite the increasing complexity of software. This entails empowering customers by improving self-help features and improving the overall diagnostic capability of Product Support in order to resolve issues faster for customers. This in turn is expected to reduce overall Total Cost of Ownership for both the customer and Microsoft.

Additionally, based on the information and data collected in this section it can be observed that there is a concerted effort in moving Microsoft's products studied in this section towards automated supportability. There is a realization that supporting products with a high degree of complexity and size as well as the high degree of integration and interconnection; would be extremely difficult while maintaining a high level of QOS and low TCO, considering that 80% of the costs associated with software occur after it is shipped to the customer [10].

The results of such a realization are clearly seen in the increasing emphasis in enhancing the products to such an extent that users can be empowered to be able to figure out problems simply

by interacting with the self-help capabilities of the software product and without any other external assistance. Similarly on the Product Support side, the emphasis is on improving the diagnostic capability to solve issues faster. Finally, this is also evident in the improved co-operation between development and support teams to address ongoing supportability problems as well as incorporating support expertise into future products.

Hence, the incorporation of advanced and standardized error detection and reporting, crash analysis, self-help and proactive monitoring are all indicative of the need to move beyond the manual to at least the semi-automated phase, keeping in mind the eventual goal of achieving automated supportability in the future.

4. SUPPORTABILITY AT OTHER ENTERPRISES

4.1. Supportability at IBM

This section deals with supportability related initiatives dealing with IBM products. IBM has an initiative geared towards deploying, upgrading, configuring, tuning and maintaining IT solutions, called autonomic computing [1]. The objectives of autonomic computing are to enable IT professionals seek ways to improve their return on investment (ROI) in their IT infrastructure, by reducing the total cost of ownership (TCO) of their environments while improving the quality of service (QoS) for users. The eventual goal of autonomic computing is to create self-managing systems that manage themselves according to an administrator's goals [18]. In other words, there is a need for computing systems to take care of themselves at a higher level, eliminating much of what is done today by human monitoring, maintenance, and control [36]. The eventual goal is to reduce TCO by reducing the number of decisions the user makes. Each Self-Managing System has the following attributes:

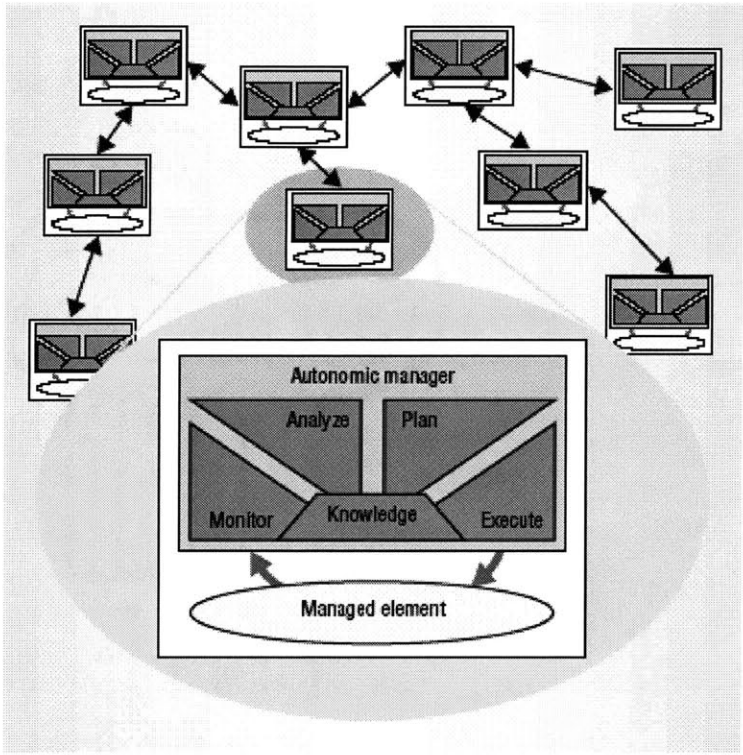
- **Self-configuration**: This essentially means that autonomic systems will configure themselves automatically in accordance with high-level policies, which represent business-level objectives, for example—that specify what is desired. When a component is introduced, it will incorporate itself seamlessly, and the rest of the system will adapt to its presence. For example, when a new component is introduced into an autonomic accounting system, it will automatically learn about and take into account the composition and configuration of the system. It will

register itself and its capabilities so that other components can either use it or modify their own behavior appropriately.

- **Self-optimization**: This attribute guarantees that autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost. They will monitor, experiment with, and tune their own parameters and will learn to make appropriate choices about keeping functions or outsourcing them. They will proactively seek to upgrade their function by finding, verifying, and applying the latest updates.
- **Self-healing**: Autonomic computing systems will detect, diagnose and repair localized problems resulting from bugs or failures in software and hardware, perhaps through a regression tester. Using knowledge about the system configuration, a problem-diagnosis component (based on a Bayesian network, for example) would analyze information from log files, possibly supplemented with data from additional monitors that it has requested. The system would then match the diagnosis against known software patches (or alert a human programmer if there are none), install the appropriate patch, and retest.
- **Self-preservation**: Autonomic systems will be self-protecting in two senses. They will defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures. They also will anticipate problems based on early reports from sensors and take steps to avoid or mitigate them.

4.1.1. Autonomic System Architecture

IBM envisions the architecture of autonomic systems comprise of interactive collections of autonomic elements. Each autonomic element is defined as an individual system that contains resources and delivers services to humans and other autonomic elements within the autonomic system. Each autonomic element interacts with other autonomic elements and manages its behavior based on rules pre-determined by humans. A distributed, service-oriented infrastructure is expected to support this architecture.



Source: "The Vision of Autonomic Computing"; Jeffery Kephart, David Chess; IEEE Computer Society; January 2003.

Figure 28: Structure of an Autonomic Element in an Autonomic System

As figure above shows, each autonomic element will typically consist of one or more managed elements coupled with a single autonomic manager that controls and represents them. The managed element will essentially be equivalent to what is found in ordinary non-autonomic systems, although it can be adapted to enable the autonomic manager to monitor and control it. The managed element could be a hardware resource, such as storage, a CPU, or a printer, or a software resource, such as a database, a directory service, or a large legacy system. The autonomic manager monitors the managed elements and makes decisions based on the monitored information.

Each autonomic element manages its own internal state and behavior and its environment comprising other elements and the world. Additionally, they can range in size from individual computing elements to small-scale computing systems to entire automated enterprises. Finally, each autonomic element is complex that continuously interacts with other elements and the environment.

4.1.2. Self-healing

The concept of supportability has a many functions that overlap with the attributes of self-managing systems, however none more so than that of self-healing. Kephart and Chess [18] mention self-healing to entail detecting, diagnosing and repairing a problem. Detection depends on how well the system configuration is comprehended, diagnosis can be performed using techniques such as Bayesian networks to analyze data followed by a repair plan that describes

the changes to make in order to recover from the problem. To repair the problem dynamically, the software architecture needs posses the following four capabilities: describe the current architecture accurately, build a clear repair plan that describes the plan, analyze the result of the change and finally implement the repair plan. Dashofy et al [37] describe an event based architecture that contains the four capabilities used xADL 2.0, an extensible, XML-based architecture description language.

4.2. Supportability at a company that develops Printer software

This section covers the supportability story at a company that manufactures printer hardware and software. Printers in general have matured over the years and some possess software supportability features that have not yet been incorporated even in commercial PC software. In this section we will concentrate on how printer software in general has managed to keep ahead of the supportability demands of today.

4.2.1. Supportability Objective

In the printer software lifecycle supportability comes into the picture once it has been released to the customer area or site. If the customer now encounters a problem then they would need to contact Product Support who would have to work towards finding a resolution. There are many means by which the problem can be resolved such as fixing the software itself if the problem lies in the printer software, providing a workaround or educating the customer to use the software

better in case of a customer error. Sometimes the customer may not encounter a problem however they may provide feedback in the form of recommendations also called a wish-list. This list comprises changes to the software in order to make it function better for them. Printer research and development might consider including such enhancements in future releases of the same version of the software system called Service Packs or in future release of a new version of the software system. Others aspects of supportability involve means used to determine the resolution itself such as using remote diagnostics to handle "after-the-fact" supportability issues and a help system for customers to refer to while operating the software system.

4.2.2. Maintenance Index

At this company there is definite awareness that maintainability of the printer source code is a very important component of supportability. In order to address source code maintainability a metric called a "Maintenance Index" [8] is compiled for all software. This index is compiled using McCabe's Cyclomatic Complexity Metrics, Halstead Complexity Metrics, Average count of Lines of code per module, and / or average percent of lines of comments per module.

Maintainability Index helps in quantitatively determining program comprehension. In other words, it helps determine how easy it is to fix the software. Its main strength lies in trying to make the source code easier to understand, hence affording the ability for better maintenance of the code base. An extreme example where the Maintenance Index helps in evaluating is in

making the decision of re-writing a portion of the source code if the code causes many problems or bugs during maintenance.

Despite its advantages it has many drawbacks. Firstly, it may or may not directly translate to better maintenance and supportability to the customer. In other words even if the source code is easier to maintain it does not generally guarantee that it may not generate supportability issues for the customer. Secondly, it does not help doing preventive maintenance, which entails measures to avoid maintenance issues from arising altogether.

4.2.3. "After-the–fact" supportability

Currently this falls under three categories. The first category involves developing troubleshooting tools. These tools offer capabilities for remote diagnosis, analyzing dumps etc. The second involves error message cataloging. This catalog comprises of a list of error messages that contain enough information for a customer to comprehend what the problem is and possible resolutions to fix the problem. Creating a meaningful catalog is generally compiled and revised based on feedback from engineers, support professionals and Beta customers. The third category comprises developing the software to generate Trace Logs. These logs output information that provides clues as what the software is /was computing when the problem occurred.

4.2.4. FURPS

A metric used conceived at Hewlett-Packard (HP) that is used at design time, which includes measurement of supportability. This metric is called FURPS an acronym for Functionality, Usability, Reliability, Performance and Supportability. Supportability generally entails issues such as Maintainability, Testability, Compatibility, Configurability, Ease of installation and Ease of problem localization.

Right at the conception of a software system quantitative FURPS goals are set that need to be met during the lifecycle of the software. HP subscribes to the idea of using cross-functionality teams for developing products. The structure of such a team allows for the inclusion of a 'Supportability Manager' and at very phase of the software development lifecycle there is a checkpoint where the product is reevaluated. This re-evaluation includes checking whether FURPS goals are being met, how the features are progressing, whether the timeline is accurate etc. The focus of the supportability or support manager to ensure supportability goals are met at each of the checkpoints.

4.2.5. Shortchanging supportability

Despite the existence of the Maintainability Index, FURPS, supportability manager, checkpoints and Recoverability issues it is not uncommon to rescind supportability features or goal during the development lifecycle or prior to ship. In fact supportability features maybe the first to be sacrificed in order to ship the product within time deadlines and with the features perceived to be what the customer requires.

There are many reasons or misconceptions that breed such a mindset. Firstly, supportability features are not what customers generally state in their requirements. Customers generally tend to request features like; make the user-interface more user friendly, or make it easier to administer, or allow it to be compatible with Product A, B or C etc. Such requirements do not explicitly require supportability features even at the detailed level. Hence, the development or design team may not perceive such features as important to the final product. Secondly, supportability features are generally 'invisible'. Customers in general are not even aware of such functionality until they encounter a problem. Hence, the customers themselves may not consider such functionality during requirement analysis. This also applies to the development or design team might consider such functionality to be expendable while make resource/ time / feature decisions in the 'Trade-off' Triangle [9]. Thirdly supportability features are perceived to be useful only when the software application has a problem. As a result they are thought to have limited scope in terms of the actual working of the product itself because the product works 'most of the time'.

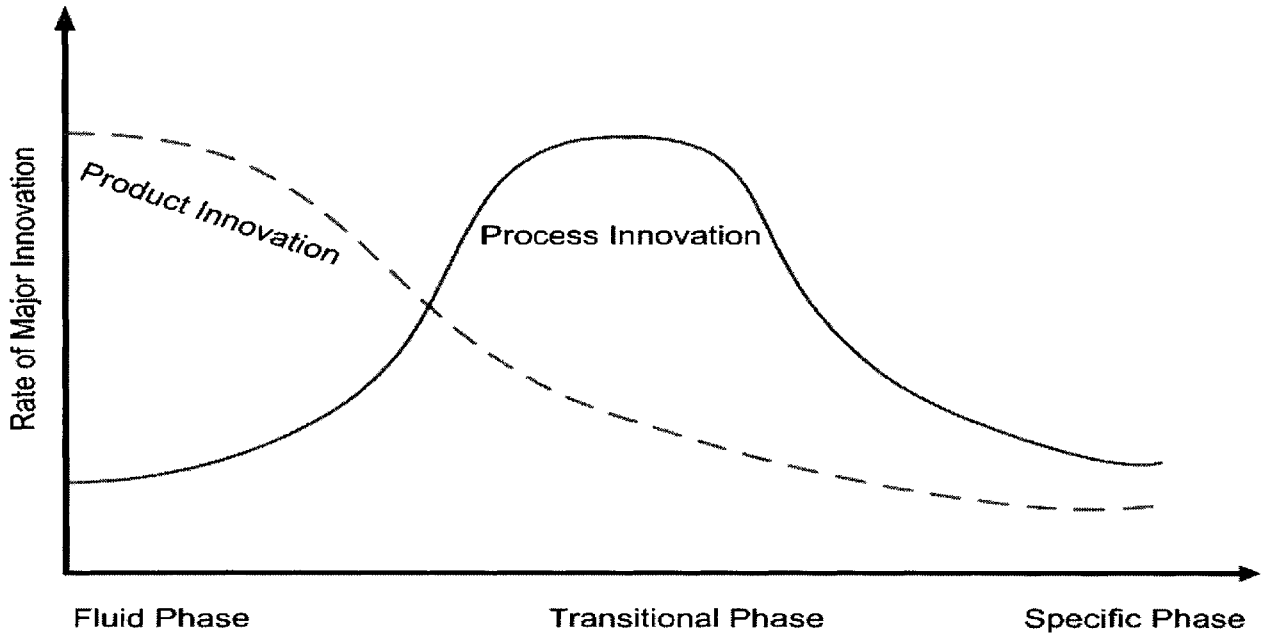
5. CONCLUSIONS AND FUTURE WORK

Engineering large software-intensive systems is a complex activity. In addition to this complexity is the software crisis itself, which encompasses amongst others a maintenance crisis [23], and a complexity crisis. Programming language innovations have extended the size and complexity of systems that architects can design, but relying solely on further innovations in programming methods will not get us through the present complexity crisis [18]. As systems become more complex, interconnected and diverse, architects and designers are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. This could be catastrophic for software systems of the near future where there will be no way to make timely, decisive, and cost effective responses to customer problems.

5.1. Software Innovation

The model for the dynamics of innovation [28] provides us with the insight that once a dominant design emerges for a product radical product innovation eventually ends and the focus of research and development focuses on incremental improvements on existing features. This is amply evident in the advent of the DC-3 aircraft, the Ford Model T automobile, the Underwood Model 5 typewriter and the IBM Personal Computer. The same seems to have emerged for

traditional software applications such as operating systems, database servers, web servers, office software etc.



Source: "Dynamics of Innovation"; James M. Utterback; p91.

Figure 29: The Dynamics of Innovation

Evidence in support of such a dominant design being achieved in traditional software applications can be seen from the products themselves. The core characteristics of the software applications themselves have not changed radically, however there continues to be innovation in the form of more features and functionality being added to the core. Another key indicator of a dominant design being reached can be found in the competitive environment that has only four major players in the Operating System market such as Windows, Linux, UNIX and Macintosh. Similarly, database servers have four major players Oracle, IBM (DB2 and Informix), Microsoft

SQL Server, and MySQL. Additionally, web server applications have Apache, IIS, SunONE, and Zeus and finally office software has Microsoft Office. Another indicator of the dominant design begin reached is the increase in activity of process innovation. The development of software engineering itself is an example of software process innovation, since software engineering itself is defined as the systematic design and development of software products and the management of the software process [32]. This innovation can be further extended by viewing examples on the introduction and incorporation of software development process improvement measures such the Software Engineering Institute's Capability Maturity Model [29]. In the software community there is still a heated debate as to the importance of the process innovations such as CMM [30] [31]; however there is no discounting the fact that more software development organizations are implementing such process innovations than ever before. Additionally, market characteristics change based on the competitive environment and the increased emphasis on more process related innovations results in important of product pricing and quality. In other words, better Quality of Service (QOS) with reduced TCO for both the customer and the software organization.

Despite the fact that a dominant design may have been achieved in traditional software applications does not imply software product innovation is done. In fact, according to Utterback [28] most technology-based innovations are in fact part of a continuum of change. In other words, there could be multiple waves of such product and process innovations that a particular technology can create.

Applying this rationale to software we can deduce the first wave of software product and process innovation to be the traditional software applications such as operating systems, database servers, web servers, office software etc. This wave seems to have at least reached the Transitional Phase in its innovation dynamics. The second wave of software product and process innovation seems to be largely related to expanding applicability of software to automate just about anything and everything such as PDAs, cell phones, video games, smart devices etc. The added complexity to all such pervasiveness of software is the need to keep all interconnected and integrated. This raises the question, 'how to prevent the dream of pervasive computing from turning into a nightmare [18]?' In other words, how will all such complex and interconnected software be supported while keeping high QoS and low TCO in mind. These three waves of software innovation are represented in the figure below:

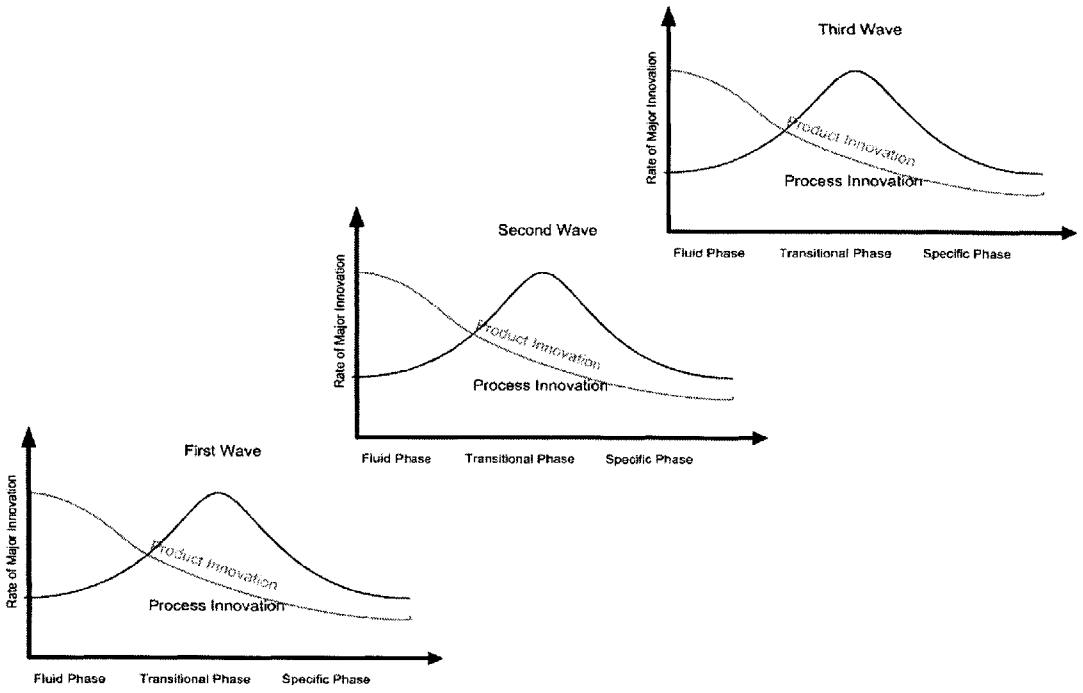


Figure 30: Waves of software Innovation

This will truly complete the launch software systems into the domain of all pervasive computing where once software is installed into its environments it simply "supports itself" when a problem occurs. End users may need interactions with the software especially in the case of pre-determined rules where explicit human intervention is required however in general most problems may not even require end user interaction. This implies the need for more software innovation, namely that of engineering fully-automated self-supporting software that essentially detects, diagnoses and fixes an anomaly or problem when it surfaces. All these three waves of software innovation can also be viewed as stacked software product lifecycles illustrated in the figure below:

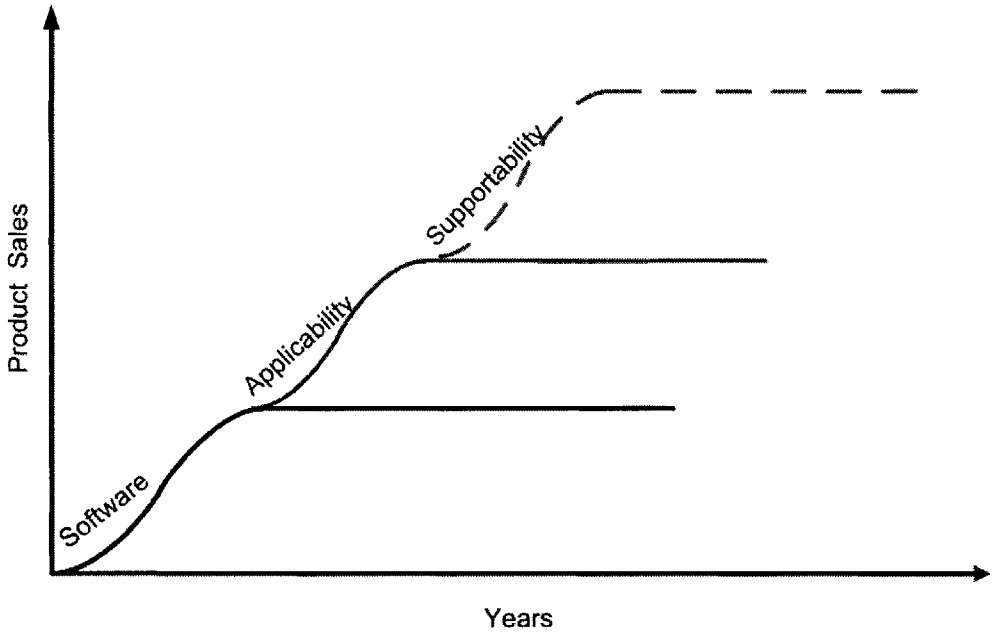


Figure 31: Future software product lifecycle

This essentially implies that software supportability's product innovation is in its Fluid Phase. This is evident from the product innovation in this area in the products observed in this thesis where definite progress is noticed in moving from manual towards semi-automated software support systems and from the self-healing feature of autonomic computing launched by IBM [1]. In essence the overall goal remains to design and develop software that essentially supports itself. In other words software that detects, diagnoses and fixes all problems that can be encountered by a software application. Generally the customer and end-user will have minimal (if any) realization of a problem.

5.2. Systems Engineering

To address this growing complex problem, which not only includes the inherent complexity of software but also the complexity of interconnecting and integrating all pervasive software requires the application of a holistic systems engineering approach rather than simply a software engineering one [34]. Systems Engineering concepts were initially coined in the 1900s in the communications and aircraft industries and then to tackle the complexity brought about by the development of strategic ballistic missiles in the 1950s [33]. It is an engineering discipline that addresses requirements analysis, design and development of complex products. A system is defined as a set of different elements so connected or related as to perform a unique function not performable by the elements alone [33] and complex is defined in Webster as composed of

interconnected and interwoven parts. System complexity stems from many sources and assessing such complexity and its sources is important in customizing the systems engineering approach to tackle it

In general, complexity can arise from the product, culture and organization. Complexity is an essential property of software, [20] hence software products are fraught with complexity. Complexity increases with the size of software code and also from the need to software to constantly evolve either due to the changing needs of its developers or customers or due to the changing of the environment it operates in case of open systems. Hence in order to design supportability into complex software systems the architecture needs to address the complexity and changeability in its architecture.

The essence of systems architecting is that form follows function. Hence is this case the function is addressing supportability demands of software customers and the form is supportable software. The demand for precisely delivering software support in time, of high quality and within budgetary constraints has never been greater. Although software support actually occurs during the post-deployment phase, it must be planned for upfront during requirements definition and design, since most of the problems associated with software support can be directly traced back to deficiencies in the way the original software product was planned, managed, and designed [23]. Additionally, considering the failure rate curve [Section 2.7] of software change there is a need to continuously address it throughout the software system's life. Hence, to ensure software success the vision should transform from developing software to developing

supportable software. Finally a flexible, modular architecture is also essential for ensuring understandability, modifiability, interoperability, reusability, expandability, and portability — all prerequisites for supportable software [23].

Cultural complexity results from the issues such as perspectives to a product, development and testing approaches, individual or team approach etc. Organizational complexity encompasses whether the organizational structure fosters proper communication and team balance to achieve the appropriate goal. We will be covering each complexity in the next sections.

5.3. Product Issues

In trying to keep up with economies of scale it is getting more difficult to support large numbers of customers in a timely and cost effective manner. In the experience of the U.S. Air force [23] systems that are difficult to support require considerable time and funds necessary to provide the required support or they had to abandon such products altogether. Additionally, it is far more cost-effective to address supportability while defining requirements, designing the system, and planning for its operational life. The commercial industry needs to innovate by learning from such experiences.

5.3.1. Incorporating supportability into software

This involves addressing complexity by decomposing the system into modules [27] and changeability by creating a flexible architecture that can accommodate changes. Additionally,

supportability related software generally requires three Atomic steps that lead to towards recovery or solving the problem. These entail the steps of detection, diagnosis and recovery. The first step is detecting the problem. This generally entails the customer or end user becoming aware that they have a problem. Depending on how much support has been incorporated into the software the customer can become aware of 'What?' the problem is as well as 'Why?' the problem is occurring. This is the most critical step towards successfully resolving the problem, however most contemporary software systems are not adept at detecting problems. As a result Product Support Services can spend an inordinate amount of time iterating complex troubleshooting steps with the customer before detecting the 'What' and 'Why' of the problem.

This step requires a first phase that entails a clear and accurate understanding of what the entire system was intended to perform. This includes design documentation, development source code comments, design of error and exception related issues such as codes, messages, detailed descriptions, call stacks etc. Subsequently it entails determining the existence of a problem when the system deviates from its intent. This includes many of the software functionality such as error code and message generation, source code tracing and logging, error and exception handling etc. In other words, it is in this step the customer or the support personal tries to make sense of the information provided by the software when the problem occurred. Sometimes the error can be as cryptic as an "An internal failure has occurred" (IBM PC-DOS2.10) or more detailed as explained by Kemal Efe [24] such as "Buffer overflow, cannot complete the creation of file 173100186.TEMP. Reset buffer to 256 records." The latter provide a lot more detailed information however an end user that does not have a detailed knowledge about the working of

the software nor have a degree in computer engineering may not know what is a 'Buffer overflow', or why 173100186.TEMP is being created in the first place, or does one need to reset the buffer. Such errors are still not uncommon in contemporary software.

Lewis and Norman [25] advocate addressing error by providing a good system design that recognizes when errors are likely to occur, minimize their likelihood of occurrence and accommodate for recovery. Additional means to address errors include explanatory user tools and providing the appropriate information to the appropriate end user. Many software development groups work closely with some of their End users, Lead Users to come up with a compiled list of errors and appropriate documentation, as in the case of printer development [Section 4.2.3]. In their paper the authors also provide different means to respond to an error such as prevent the behavior that caused the error to continue, gag, warn, Do Nothing, Self Correct, Teach Me, and Let's Talk about it. Microsoft in their self-help software development incorporating more of the "Let's Talk About It" (LAIT) response. This enables the software to initiate a dialog with the customer. This helps facilitate not only improved error detection but also leads onto better diagnosis with possible recovery alternatives as well.

Another aspect of detection is proactive monitoring of the software application at regular intervals. In this case many metrics of the software application are gathered at regular intervals and the data gathered is analyzed and evaluated against existing rules and boundary conditions to determine whether the software is function doing what it was intended to do. If not then it takes appropriate action to ensure a timely and graceful recovery. An example where external software products monitor other software is the Microsoft Operations Management (MOM) infrastructure.

Sometimes the software itself detects certain abnormal condition and takes measures to recover from it. An example of this can be seen in Microsoft SQL Server detects existing database size limitation and instead of throwing an error it proactively increases its size to accommodate for this change.

The second phase is that of diagnosis, sometimes called debugging depending on the complexity of the problem in order to get resolution and the software. The second phase addresses any lingering questions on 'Why' and the 'How' to resolve the problem aspect of the problem. It is important that the first step feeds in correct and accurate information to this phase else a lot of time can be wasted in providing the diagnosis of a completely unrelated problem. In any case, most contemporary software relies on the diagnosis skills of their support personal. Automating this step is extremely complicated since debugging prowess appears to be an innate human trait [10] and one of the most frustrating parts of programming [26]. As mentioned in the previous step the LAIT error response can be applied to take diagnose and take corrective action. For more complicated scenarios where dialog with the end user may not suffice in determining root cause, then automation of human debugging techniques using advanced tools becomes important. Examples on such tools could be on-line automated crash analysis or error report analysis tools. Microsoft provides an online crash analysis tool, which is able to automate diagnosis of some common crash scenarios. Another example of automated diagnosis is a Bayesian belief network shown in Section [12].

The final step of the solving the problem lies in actually providing the solution or recovery from the error. Once again this step is heavily dependent on the first and step because providing a fix to an incorrect diagnosis and detection will still not solve the problem. Sometimes the system provides in-built design capability to correct the problem itself, which is analogous to the Self correct response. [25] Additionally, the extension of the LAIT response could end with the corrective action which could be installing an existing fix or implementing a workaround etc. This is one of the alternatives that the self-help functionality in Microsoft software tries to achieve.

5.3.2. Classification of software supportability

Based on how well the software design incorporates these steps into its design it falls under the classification presented in Section 2.8:

- **Manual supportability**: Manual software support depends entirely on manual intervention by either software support services or software development or both for detecting, diagnosing and recovering. Most contemporary software systems fall under this category.
- **Semi-Automated supportability**: Semi-automated software support would provide some degree of automation in one or all of three troubleshooting phases. In other words, the software is automated to support some anomalies or parts (detecting, diagnosing or correcting) of anomalies however need human intervention to diagnose and correct all anomalies. The detection Phase generally entails features such as detailed Tracing or logging, error detection, or monitoring.

In the diagnosis phase this could be automated crash analysis or error analysis services that would provide answers as to how to go about fixing the problem. Finally, for the Fixing phase this could involve a self-fixing feature that could provide a fix to the problem based on the diagnosis.

- **Automated supportability**: Systems that do not depend on any human intervention to detect diagnose or fix problems. Fully-Automated supportability would essentially automate each of the three troubleshooting phases within the software itself.

5.3.3. Measuring supportability

In order to improve supportability there needs to be a method to measure it during its design phase as well as during its support phase. Such measures when tend to shape the design by explicitly specifying 'What' needs to be incorporated to make the software supportable.

- The first important factor that affects software support is the inherent complexity of software [20]. Complexity of software itself can be reduced by decomposing the system into modules [27]. Hence a good metric used is measuring the complexity of individual modules. There many software metrics to measure complexity, however one generally accepted complexity measure was found to be McCabe's Cyclomatic Complexity Measure. The Air Force Guide Specification (AFGS) [23] generally places this to be no more than 10 for a given module.

Another measure of software complexity that is also widely used is Halstead's Metrics and can be found in Halstead's theory of software science.

- Another factor that affects software support is the size of the code that needs to be supported. Once again metrics used to determine this factor are based of the size of individual modules. Generally the size-oriented metric used is source lines of codes (SLOC) or thousands of lines of code (KLOC). The Air Force Guide Specification (AFGS) [23] generally places this to be no more than 100 SLOC for a given module.
- The next important factor to note is that software support varies depending on the programming language used to write the software. There have been improvements in contemporary high-level programming languages that reduced complexity and improved productivity of programmers to write more lines of code. This has a significant impact on software support. Additionally, all the previously mentioned metrics vary as well depending on the programming language. Hence, the choice of the programming language itself plays a very important role in either enhancing or degrading supportability depending on the software application.
- The software entity is constantly subject to pressures for change [20]. Such software changes are classified as enhancements or refinements. In order to address the high failure rates resulting from new defects getting introduced or latent defects getting uncovered when such change is incorporated into the software [Section 2.7] there is a need to foresee and accommodate for such change in the design. Such change can be in the form of amount of memory, number of processors, number of users, number of devices etc.

- An important metric to measure effectiveness of supportability measures is the amount of time and resource expended in trying to resolve or fix a customer problem. This measure can then be used to compare time and resource expended to solve a similar problem before the supportability features may have been introduced into the software. A metric proposed in this thesis is called the Supportability Factor [Section 3].

5.4. Cultural Issues

It goes without saying that product related technical issues are supposed to be the easy issues compared to cultural or people related issues. This argument holds water here as well. The shift from simply designing software to designing supportable software involves a change in perspective and approach to designing and developing software. This increases the complexity in terms of building such software products. Contemporary software product designers and developers are constantly striving to incorporate and automate as many features into their respective software products to increase applicability of software, as elucidated in Section 5.1. In order to incorporate supportability into their software not only do designers, developers and testers need to take features into consideration but also consider how they need to make each feature supportable. Injecting this capability is complex in itself which adds to the overall complexity of the software product.

Additionally, supportability features are not visible when the application doing what it was intended to perform. They are noticed when the application does something that it was not intended to do. Hence, in today's world where all features need to be developed by 'yesterday', designers and developers are tempted to overlook such issues in order to ship the features that the customer will notice. Hence a probable scenario could that under time and resource constraints to ship the features, its supportability related features may be condoned while making a ship or not to ship decision. This short term perspective needs to change to incorporate the long term vision of QOS and TCO over the entire lifetime of the product. There is a need to re-iterate the statistics that according to numerous DoD and industry studies, the typical cost to simply maintain a software product is from 60% to 80% of total life cycle costs. [23] In general, it is found that nearly 80% of the costs associated with software occur after it is shipped to the customer. [10]

5.5. Organizational Issues

Organizational complexity issues deal with whether the organizational structure fosters proper communication and team balance to achieve the appropriate goal, in this case the goal is to design supportability into software. It is very important get the right balance of product and functional teams in order to reduce organizational complexity.

Depending on the function that the product needs to provide, the design or project team in a matrix organization can comprise appropriate cross-functional members. In other words, borrowing from experience in the aerospace we can see that Design teams are characterized by multi-discipline teams comprising of representatives of Engineering, Product Support, and

Manufacturing. [35]. The use of multi-discipline teams has been practiced in other forms of engineering civil, bio-medical etc. Furthermore, this can be applied to software as well; where in order to incorporate supportability into software there could be a Product Support personal involved in the design team of the product. In order to address complexity of the software each design team can own a module of the software obtained by decomposing the overall complex product. This can be called a centralized approach, where each module of the software contains multi-disciplined design teams with at least one member solely dedicated to supportability.

Additionally, a design team itself can own the discipline of supportability while belonging to a design team. This can be called a decentralized approach, where each module of the software contains design teams where one or more member keep supportability issues in mind while designing and developing the product. The drawback generally is that unless the team member is committed to the discipline there is a high-likelihood of it getting shortchanged during the design and development of the software product.

5.6. The supportability vision

The supportability vision encompasses all the issues (the what) discussed in this section. In order to completely achieve the thesis of "Designing SUPPORTABILTY into Software" we need to achieve fully automated self-supporting software systems. Hence, in order to achieve this vision we would need to apply sound system engineering concepts to deals with product, cultural and organizational complexity. Hence the complete vision could entail:

"Engineer automated self-supporting software systems by applying sound system engineering concepts to deals with product, cultural and organizational complexity to achieve lower TCO and higher QoS."

5.7. Future Work

Software innovation leads up the path to creating automated software support [Section 5.1]. As mentioned engineering automated self-supporting software systems is a very complex undertaking that is currently in its initial stages where most software systems currently employ either completely manual or mostly manual software with some semi-automated support systems.

Software products at Microsoft are making the transition towards semi-automated and then towards automated software support systems. Additionally, other big software companies like IBM are also working towards autonomic systems, which contain the trait of self-healing in which the software determines the root cause of failures in complex computing scenarios.

The transition to automated software support cannot occur overnight. Incremental gains and lessons learned in intermediate or semi-automated software support systems will provides realizations into the existence of such completely automated support systems that 'simply support themselves'. These gains encompass those in decision theory, detection, monitoring,

diagnosis, recovery, fault-tolerance etc. Additionally gains need to be made in dealing with people and organizational issues to ensure that a holistic systems engineering approach is taken to tackle supportability. This thesis provides a closer look on 'What?' supportable software entails and the application of systems engineering to tackle such a complex issue. The incremental gains while engineering such systems will provide us insights as to 'How?' complex software systems can actually be engineered.

These incremental gains will help us determine how to improve the end-users overall QOS, designing better detection mechanisms such as errors, exceptions, tracing etc., better decision making to determine user intent by using various techniques such as Bayesian Probability, Artificial Intelligence etc, boundaries as to when human intervention or awareness is required, elements of fault-tolerance in order for the software systems to make a successful recovery, etc.

REFERENCES

[1] "Autonomic Computing"; IBM; http://www.research.ibm.com/autonomic/overview/; Accessed July 2003.

[2] "An architectural blueprint for autonomic computing"; IBM; http://www-3.ibm.com/autonomic/pdfs/ACwpFinal.pdf; Accessed July 2003.

[3] "Software Complexity and Maintenance costs"; Banker, Rajiv et al; Communications of the ACM; Vol. 36 No. 11; November 1993.

[4] "Strategic Directions in Software Quality"; Osterweil, Leon; ACM Computing Surveys, Vol. 28, No. 4, December 1996.

[5] "Automated Design Decision Support System"; Beggs, Robert; 29th ACM/IEEE Design Automation Conference, 1992.

[6] "Programming without tears"; Freedman, D.H; High Tech; 6, 4 (1986); 38-45.

[7] "Software Maintenance Costs: A Quantitative Evaluation"; Herrin, William R; ACM SIGCSE Bulletin, Proceedings of the sixteenth SIGCSE technical symposium on Computer science education; Volume 17 Issue 1; March 1985.

[8] Maintainability Index Technique for Measuring Program Maintainability; http://www.sei.cmu.edu/str/descriptions/mitmpm_body.html

[9] "Rapid Development"; Steve McConnell.

[10] "Software Engineering a Practitioner's Approach"; Roger S. Pressman.

[11] AutoAdmin: Self-Tuning and Self-Administering Databases; http://research.microsoft.com/dmx/autoadmin/default.htm

[12] MSBNx: Diagnosis and Troubleshooting; Help Contents and http://research.microsoft.com/adapt/MSBNx/Background.asp

[13] "Persistent-state Checkpoint Comparison for Troubleshooting Configuration Failures"; Wang,Yi-Min; Verbowski, Chad; Simon, Daniel R.; International Conference on Dependable Systems and Networks, 2003.

[14] Microsoft Operations Manager; http://www.microsoft.com/mom/

[15] "EXPLOIT the Product Life Cycle"; Theodore Levitt; Harvard Business Review; Nov/Dec65, Vol. 43 Issue 6, p81, 14p

[16] Detours; http://research.microsoft.com/sn/detours/

[17] Microsoft Online Crash Analysis; https://oca.microsoft.com

[18] "The Vision of Autonomic Computing"; Jeffery Kephart, David Chess; IEEE Computer Society; January 2003.

[19] "Splitting the Difference: The Historical Necessity of Synthesis in Software Engineering"; Stuart Shapiro; IEEE Annals of the History of Computing; Vol. 19, No. 1; 1997.

[20] "No Silver Bullet: Essence and Accidents of Software Engineering"; Frederick P. Brooks, Jr.; Computer, vol. 20, p. 12, Apr. 1987.

[21] ANSI/IEEE Standard 729-1983, IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronics Engineers, Inc., New York, 1983.

[22] "Software Document Evaluation Guidelines for DOD-STD-2167A"; Faye Budlong, Reed Sorensen; Crosstalk, June, 1996.

[23] "Guidelines for Successful Acquisition and Management of Software Intensive Systems", Version 2.0 June 1996.

[24] A proposed solution to the problem of levels in error-message generation; Kemal Efe; Communications of the ACM; Volume 30 Issue 11; November 1987.

[25] "Designing for error. In User Centered System Design"; Lewis, C. and Norman D.A.; Eds. Erlbaum. Hillsdale. N.J., 1966.

[26] "Software Psychology"; Shneiderman B; Winthrop Publishers, 1980, p. 28.

[27] "On the Criteria To Be Used in Decomposing Systems into Modules"; D.L. Parnas; Communications of the ACM; Vol. 15, No. 12; December 1972; pp. 1053 - 1058.

[28] "Mastering the Dynamics of Innovation"; James M. Utterback; Harvard Business School Press, Chap. 4; pg 79 - 102.

[29] "The Capability Maturity Model for Software"; Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Charles V. Weber; CMM v1.1.

[30] "Enough About Process: What we Need are Heroes"; James Bach; IEEE Software; March 1995.

[31] "The Immaturity of CMM"; James Bach; American Programmer; September 1994.

[32] "The management of software engineering Part I: Principles of software engineering"; H. D. Mills; IBM Systems Journal; Vol. 38, NOS 2&3; 1999.

[33] "The Art of Systems Architecting"; Mark Maier, Eberhardt Rechtin; Second Edition.

[34] "Systems Engineering: An Essential Discipline for the 21st Century"; Donna H. Rhodes; ACM SIGSOFT; September 2002.

[35] "Automated Design Decision Support System"; Robert Beggs, John Sawaya; 29th ACM/IEEE Design Automation Conference; 1992.

[36] "Managing with ghosts: Managing the user experience of autonomic computing"; D. M. Russell, P. P. Maglio, R. Dordick, C. Netti; IBM Systems Journal; Vol. 42, No. 1; 2003.

[37] "Towards Architecture-based Self-Healing Systems"; Eric M. Dashofy, Andre van der Hoek, Richard N. Taylor;

[38] "Microsoft Windows XP System Restore"; Bobbie Harder; April 2001; http://msdn.microsoft.com/library/default.asp?URL=/library/techart/windowsxpsystemrestore.htm.

[39] Reliability Improvements in Windows XP Professional; MSDN; July 2001; http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/winxppro/Plan/RlbWinXP.asp.

[40] "Lean Thinking"; James P. Womack, Daniel T. Jones; 2003.

APPENDIX: GLOSSARY OF TERMS

Add-in: Add-in is a term used, especially by Microsoft, for a software utility or other program that can be added to a primary program. [Source: SearchVB.com]

Bug Fix Release: A release which introduces no new features, but which merely aims to fix bugs in previous releases. All too commonly new bugs are introduced at the same time. [Source: Hyperdictionary.com]

Core Dump: Also referred to as a dump is the printing or the copying to a more permanent medium (such as a hard disk) the contents of random access memory (RAM) at one moment in time. One can think of it as a full-length "snapshot" of RAM. A core dump is taken mainly for the purpose of debugging a program. [Source: Whatis.com]

Crash: It is a sudden failure of an application or operating system or a hardware device that results in its abrupt and abnormal termination.

DNS: It is an acronym for Domain Name System. The domain name system is the way that Internet domain names are located and translated into Internet Protocol addresses. A domain name is a meaningful and easy-to-remember "handle" for an Internet address.

DoD: It is an acronym for the Department of Defense.

Dump: (See Core Dump)

Knowledge Base: In general, a knowledge base is a centralized repository for information: a public library, a database of related information about a particular subject, and whatis.com could all be considered to be examples of knowledge bases. In relation to information technology (IT), a knowledge base is a machine-readable resource for the dissemination of information, generally online or with the capacity to be put online. [Source: SearchVB.com]

Major Release: A release of a piece of software which is not merely a revision or a bug fix release but which contains substantial changes (e.g., an overhaul of the interface, change in compatibility). Traditionally, major releases are numbered as X.0; for example, WordPerfect 6.0 is a major release, significantly different from any previous version; whereas WordPerfect 6.1 has only minor changes, and is, thus, only a revision. [Source: Hyperdictionary.com]

QFE: Quick Fix Engineering (QFE) is a Microsoft term for the delivery of individual service updates to its operating systems and application programs such as Word. Formerly called a hotfix, "QFE" can be used to describe both the method of delivering and applying a patch or fix, and also to refer to any individual fix. Because of the complexity and sheer number of lines of code in most application programs and operating systems, the delivery of temporary fixes to users has long been provided by major software manufacturers. Typically, not all fixes are necessarily applied by an enterprise since they can occasionally introduce new problems. All of the fixes in any given system are usually incorporated (so they don't have to be reapplied)

whenever a new version of a program or operating system comes out. Periodically, all current QFEs (or hotfixes) are delivered together as a service pack, which can be applied more efficiently than applying fixes one at a time. [Source: Whatis.com]

Service Pack: A service pack is an orderable or downloadable update to a customer's software that fixes existing problems and, in some cases, delivers product enhancements. [Source: SearchVB.com]

Wish-list: A list of features that the customer wants, but the software does not currently implement.

Workaround: A workaround is a method, sometimes used temporarily, for achieving a task or goal when the usual or planned method isn't working. In information technology, a workaround is often used to overcome hardware, programming, or communication problems. [Source: SearchVB.com]