Automating Software Failure Reporting

There are many ways to measure quality before and after software is released. But for commercial and internal-use-only products the most important measurement is the user’s perception of product quality. Unfortunately perception is a very difficult thing to measure so companies attempt to quantify it through running customer satisfaction surveys and through the collection of failure/behavioural data from its customer base. This article focuses in on the problems of capturing failure data from customer sites. And while we’ll be using the experienced gained from collecting failure data from Windows XP systems to explore the pertinent issues, the problems one is likely to face when developing internal (non-commercial) software should not be unsimilar.

A Little Historical Perspective

Traditionally computer companies collected failure data through their customer’s or their own service arm, manually submitting bug reports. Back in the 70’s and 80’s a number of computer companies (IBM, Tandem, and Digital etc) started to service their customer’s computers through electronic communication (usually a secure telephone link). It was a natural progression to automate the collection of failure data, whereby whenever the computer/application crashed its failure data is automatically collected and sent back to the manufacturer, where it can be forwarded to the engineering departments for analysis.

The initial focus of these processes were to address those failures that occurred due to the product not performing its ‘advertised’ functionality (i.e. system crashes due to software bugs). But, as Jim Gray’s analysis of the failure occurring on Tandem systems identified in the late 70’s, a large proportion of customer failures occur as a result of user actions, often categorized as HCI failures (Human Computer Interaction). This is in spite of the Tandem systems being managed and serviced by highly trained personnel. The root causes of HCI and complex failures (such as those caused by the incorrect re-configuration of a system) are difficult to diagnose, especially when the collection processes is primarily focused on failure data such as crash dumps. As such these processes (and the internal fault management system) had to evolve to improve the ability of engineers to diagnose the causes of all system failures.

Prior to the late 90’s the traditional methods of collecting failure data was dependent upon the company having their own service arm and developing and maintaining a means of communicating with their users. The evolution of the internet provides a mechanism for software producers of any size to have an affordable method of communication with their users.

It seems obvious that all companies should develop a process to collect customer failures and to distribute patches to fix any problems. Such a process benefits both the software producer and the end user. Unfortunately a badly thought out process can produce vast amounts of data which cannot be analysed and at the same time alienate the customer base. The following tale is an excellent example of how things can go wrong.

Digital Equipment Corporation (DEC) wanted a better understanding of why system managers were rebooting their systems (for every one crash a system would have 10 reboots). The why boot process asked the system manager, during system reboot, the reason for their action. The response was captured in the system event log that DEC subsequently collected. When the process was rolled out to a few sites the problems became evident.

Most of their customers set their servers to automatic reboot, which while not perfect does often resolve issues (if the failure occurs due to the system running out of resources, due to software
leaking or ageing, then a reboot will free up those resources, at least for a short period of time thereby allowing the users to utilize the computer). Automatic reboots are particularly useful when the system manager is not available 24x7. The why boot process stopped the system from rebooting as it required an input from the system manager. This resulted in long outages. Even when the system manager was present why boot still caused problems. Installing a new application on a cluster may require reboots to a number of computers in sequence. During every reboot the System Manager would have to wait at the console to answer the why boot question. Naturally the System Managers were not inclined to answer the why boot questions with any great accuracy (usually the field was blank or filled with cryptic or not very polite comments).

The why boot program while being developed by an experienced computer manufacturer, is an example of a process that was expensive to develop and deploy, collected non actionable data and annoyed the end customers. And thus the question: What are the issues that need to be considered to avoid such disasters?

Understanding your products user base and its usage profile

It is important to ensure that any data collected from the customer base is unbiased, or at least any bias is understood. There are a group of users who are very good at filling in bug reports; these users are usually technically competent and are willing to go through the sometime cumbersome and time consuming process of filling in these reports. While these users are invaluable they are not necessarily representative of the user base (it appears that the more the product is targeted at the home user then the less representative these users are). Therefore to ensure an unbiased data set the user interface of any process must be targeted towards the average user and should ensure

1. The user interface is as simple as possible. A single button click is about the maximum acceptable for the average user.
2. The request for data collection must occur at a time that will not annoy the user
3. The users will see the request for data; otherwise users may become distrustful of the process, possibly viewing it as a form of spy ware, and turn it off.
4. The process must respect user privacy; otherwise the users will be reluctant to provide any information.

These objectives can only be achieved if the usage profile of your product is well understood. If the product is used in a business and home environments then your process may need to adapt to the different markets. Additionally if the product can be used in a client or server environment then the interface should change based on the context of the usage. For failures occurring in a client environment the data request should be made to the current user; for a server the request must be redirected to the system manager and the data collected will have to go through an authorized path.

Defining the failure profile of your product

A product fails when it does not conform to its specification. This formal classification of failures is not really applicable in commercial products - the users don’t usually read the specification. The customer definition of a failure is that the product does not do what they expected. While this failure classification may appear to be boundless, most customers do apply common sense in using this definition. As such crashes may not be the major cause of customer dissatisfaction. The users may be more frustrated by product behaviour such as requiring a reboot to correct an action, by performance glitches or by confusing behaviour.
A second issue is the need to understand the product package. From an engineering perspective a product is bounded by its own software. Users may view things differently. For instance if a product fails to print then this may be viewed as a product defect even if the defect exists in a 3rd party print driver or operating system settings. This is especially true if all other products on the computer do successfully print.

Typical indirect product failures are:
1. User interface failures due to illogical inputs or using the software in unintended ways.
2. Using the software with components that are different from the recommended configurations.
3. Hardware failures that corrupt storage.
4. Software failures occurring in drivers or dependent 3rd party applications.

In addition a products failure profile is rarely static. New patches that fix known bugs and changes in the system configuration and hardware can all change the environment. Analysis of Windows XP failures highlights the range of possible system configurations; currently there are over 800,000 different kinds of plug and play devices on customer sites, with 1,500 devices being added every day. Additionally there are 31,000 unique drivers with 9 new drivers being added daily (each driver has approximately 3.5 versions in the field with 88 new driver versions being released daily). This is also compounded by the average customer system continually changing, whose average speed is currently increasing at approximately 5 MHz per week.

Capturing failure information

It is difficult to predict the information required to diagnose failures. You should assume that the initial data set will be insufficient to diagnose all possible failures. Therefore the process should be designed to evolve after its distribution to the end customers.

The following set of generic data helps diagnosing most product failures. Unfortunately in implementing this list the engineer must also realize that there is a practical limit to the amount of data that can be collected from the customer site (discussed later in the article).

- **Crash data** is captured in the product dump file and generated at the point of failure. As dump files can contain the total contents of the system memory it is often necessary to process the dump file to extract only the most relevant data.
- **System profile** including the version of the product and the patches. Also useful are the versions of the hardware and other applications that the product is dependent upon.
- **Failure history** is an important factor in helping diagnosing product failures, specifically what happened to the system just prior to the failure? Many product failures are induced by external events (e.g. configuration changes, failures to other parts of the system etc.).
- **User input**, while in general you should avoid any manual input as it may result in skewed or no data (as the users may get annoyed at such requests). But if, very occasionally, additional information is required, users are generally happy to provide it.

Identifying the data that needs to be captured is not a purely technical problem; another important factor is privacy.

Privacy

Privacy laws vary greatly across the world and it should be assumed that collecting personal data (without user permission) is illegal. Even if one asks for permission legal issues still apply to
the way personal data is stored and managed. Therefore for general purpose data collection, no data should be collected such that it is traceable back to the end user. If there is a need to correlate failure data against the user profiles then a different data collection process must be developed and targeted at customers who understand and accept the process.

While collecting personal data should be avoided it is essential to differentiate between multiple failures occurring on multiple systems from multiple failures on a single system. This can be achieved by building a signature based on the computer configuration. While not perfect, as configuration changes alter the signature, it appears to be the best practical solution.

**Processing and collecting failure data on the customer system**

Collecting failure data requires a process resident on the customer’s computer that detects processes and transmits the data. This can be achieved in many ways and is very dependent upon the products customer base. For Windows this process is enabled by a dialog with the user as part of the installation process. Thereafter the first time a system administrator logs onto a system following a system reboot, the operating system automatically checks if the cause of the system outage was due to a system crash. If so, it processes the system dump file generating a mini dump and an xml file that contains the versions of all drivers on the system. This data is subsequently compressed.

A prompt then appears on the screen requesting the user’s permission to send this data to Microsoft. If the user agrees, the data is sent via HTTP POST. This method also allows the process to send a response back to the user through redirecting the http request to a web page containing possible solutions (this is discussed in the next section).

If the computers exist within a corporate environment then some corporations may restrict internal computers from sending data externally to the company. This severely complicates data collection and often requires a two stage process. One process automatically routes the failure data to a central system(s) within the corporation; a second process sends this data off site. In this scenario it may be necessary to provide a second type of report to the corporations defining a list of patches recommended to be installed onto all corporate systems.

**Analysis Engine**

The failure data collected from customer sites is fed into a process which analyses, stores and if possible feeds back information to the end customer. The collection and analysis process must be completely automated. The collected data is processed and the analysis engine should use a set of rules to diagnose the cause of the failures. The analysis engine is continually updated by service and development engineers assigned to debugging failures. By categorizing and storing the collected failures it is possible to prioritize engineering effort onto the most frequently occurring bugs.

On most products a small percentage of defects result in the majority of failures on customer sites. As such the analysis process initially focuses on these failures both in finding a resolution to the defect (usually a patch) and in identifying future failures of this type. If a crash is due to a known defect then the reporting system should inform the users of the availability of a patch. This feedback mechanism encourages users to submit failure information.

In the period following its release Windows XP failures were heavily skewed. A very small number of bugs were responsible for the majority of customer failures. The analysis engine identified these crashes based on the specific place that the system crashed and what drivers were loaded on the system. The initial focus was the generation of patches with the assumption that this will result in a significant decrease in the total number of Windows XP crashes. In reality the rate of
failures due to these bugs continued to grow and that forced us to re-think our patch distribution mechanism (this is discussed in the next section).

Windows engineers then started to encounter crash categories that were not as easy to solve, and over time several strategies have been developed to help debug these failures, specifically

- Improving the quality of the data collected from customer sites.
  For example Windows XP SP2 will collect additional information with a focus on hardware (e.g. BIOS version, ECC status, processor clocking speeds to identify overclocking). As Microsoft shares failure data with partners a number of these companies now store manufacturing information on the system that is collected as part of the dump process (e.g. some manufactures store the make and date of installation of every product).

- Special tests to identify hardware failures.
  Tests have been written to identify hardware related failures. For instance as part of the crash dump, several memory pages are captured that contain operating system code. These pages are verified to see if they have become corrupted. If corrupted it is possible to identify the possible causes and recommend solutions to the customer (e.g. if the corruption is hardware related the customer is pointed to a hardware memory checker).

- Developing data mining tools to assist in failure diagnosis.
  Engineers are assigned to a group of crashes that the analysis engine believes are due to a single cause. In addition to the data in the crash dumps, tools are available for the engineer to mine the crash database for other relevant information (e.g. identifying the frequency of the combination of specific drivers in other crash grouping etc).

As the engineers resolve the causes of crashes, the analysis rules are updated to identify all future crashes of this type. The percentage of Windows XP crashes that can be automatically resolved through the analysis engine continually fluctuates. While patches are released to resolve current issues, new drivers and peripheral devices, of various qualities, continually appear. It is difficult to know what the ideal diagnosis rate should be; diagnosing a high percentage of bugs may simply indicate that the patch distribution process is broken.

Of the currently diagnosed Windows XP failures, 5% are Microsoft software bugs, 12% are due to hardware failures and 83% are due to third party failures.

The following pie charts provide a breakdown of the causes of hardware and driver crashes.
The percentage of hardware failures appears to be increasing; probably due to the ageing profile of the systems running Windows XP (new systems bought when XP was released are now 3 years old). As previously mentioned the failure ratios due to 3rd party drivers vary over time, as the release of a new version of a popular driver may result in an increase or decrease in failure rates.

Sometimes the solutions to these problems can be embarrassing. For instance analysis of many driver crashes showed that the root cause was that the drivers did not check for error conditions following a system call. In discussions with the developers, they said they copied the code from help files in Microsoft’s Drivers Development Kit and MSDN online documentation. The original documentation was written to provide succinct examples of how to use the system calls and so lacked error handling. The fix; the documentation has been rewritten to include checks for error conditions.

**Patch distribution**

The Microsoft analysis engine is designed to correct common crashes by providing a patch facility when the crash signature matches. An important issue is whether the patch should be distributed to all customers through a generic patch distribution process: thereby preventing other system crashes. The development of an effective patch distribution process is complex. Here are a few of the factors that need to be addressed to develop a successful process.

- **Patch quality** should be thoroughly tested in as many user environments as possible prior to distribution. While the inclination is to release the patch as soon as possible, especially if the patch addresses security defects: if a patch subsequently causes problems then customers will be less inclined to install future patches.
- **Versioning** so it is possible to verify what patches have been currently applied to the system. This is important where patches have dependencies to other patches as the versioning information is required to resolve these dependencies.
- **Patch size** is a problem for both sides. The smaller the size of the patch the smaller the load applied to the server to download the patch and also the less time it will take the end user to download and install the patch. A good test is whether the patch can be practically downloaded by users with 28k modems.
Number of patches per year. It may appear logical to release as many patches as possible for all known failures. Unfortunately this can place a great a load on the end users. Corporation have to test and stage each patch prior to its release. The process should only release patches for problems that affect a noticeable percentage of the users.

Installation of the patch should be through a single button click and it should have as minimal an impact on the end user as possible.

Automatic deployment for the home user. For Windows operating system Microsoft distributes patches though an automatic pull process (the process checks the Microsoft site to identify if critical patches are available and if so downloads them as a background process). This was deemed necessary to deploy security patches to as many users as possible. For deployment of bugs to non critical applications then a less invasive patch distribution is probably preferred.

Installation should be through a single button click and it should have as minimal an impact on the end user as possible.

Staged deployment for the corporate users. For ease of management some business like their employees to be running a common set of software at a common version. To control these environment corporations may prefer to distribute the patches themselves. As part of this process they will stage and test the patches to verify their correctness, within their own environments, prior to distribution.

While the patch distribution process can be the most important part of the whole process, as there is little point generating patches if no customers subsequently install them, there may be other ways to get patches installed onto customer sites. One possible method is to try to ensure computer manufactures pre-install all patches, as part of their build process.

Summary

The development of a successful bug reporting systems is hugely beneficial to both software producers and users. It can provide an excellent understanding of the quality of company’s products, while giving customers an improved user experience. But developing such a process is complex and if done incorrectly can do more harm than good in terms of customer satisfaction.

Developing a failure reporting system requires an understanding of the products customer base, and usage profile. It is also important that you understand the failure profile of your product so that you can focus on those events that most annoy your customers. While product attributes are unique there are a generic set of data that if collected will help in the diagnosing of failures. But in collecting customer data you must address all privacy concerns prior to rolling out your process. In parallel to developing a failure collection system you also require a process that can distribute patches to address those failures.

Our experience has led us to develop a generic methodology, to process, transmit, analyse and respond to customer failure data. Differences exist in the way this process can be implemented, and is usually dependent upon the product type and its failure profile.

While the development of this system has been hugely beneficial to our organization, the usage profiles of computers are continually changing, as their configurations evolve, so their failure profile will never be static. As such, no matter how successful an automatic software failure reporting process is, it will always need further development.