Ed Bryce

Ed has over twenty-one years of experience in software development, Quality Assurance, and testing in a variety of industries including manufacturing, software applications, engineering design, and systems integration. With seven years at Microsoft, Ed has been responsible for multiple-project test labs, data warehousing testing, web and traditional client-server based internal applications, localization testing, and meeting ISO 9000 requirements for European clients of Microsoft Consulting Services. Ed managed large cross-functional teams across multiple business units and was responsible for multi-million dollar budgets.

Prior to Microsoft, Ed worked at the Aldus Corporation (now Adobe) managing the testing effort for the PageMaker desktop publishing application. Ed also has experience at Deloitte and Touché working on a multi-year software integration project for The Boeing Company, where he was the Quality Assurance manager over the multi-company team.

In addition to traditional software development projects, Ed also designed, developed, tested, and implemented aerospace manufacturing systems for a major missile manufacturing company.

Reality Test

Failure Is Not an Option

Abstract

This paper discusses the factors involved in determining the cost of a 24-hour by 7 days per week (24 X 7) e-Commerce or internal web site going offline for any length of time. After determining these costs, and showing a real-life example calculation, the paper then goes into several ways to minimize this risk via hardware architecture, software architecture, and stress testing.

Presenter: Ed Bryce
Reality Test
704 228th Ave. NE #671
Sammamish, WA 98074
www.Reality-Test.com
Ebryce@Reality-Test.com

Cost of Failure

While this discussion of cost may be applied to any web site operating for any given period of time during the day, its primary focus is on web sites that are supposed to be available continuously, or 24 X 7 (24 hours a day times 7 days per week).

Additionally, there are web sites that are used internally, such as for a corporate enterprise application, or externally, for an e-Commerce application where the company hopes to make money from their web site. This paper applies consistently to web sites for either of these two purposes.

Finally, there is planned and unplanned downtime. Planned downtime may be for events such as a software upgrade, while unplanned time may be due to a power failure causing the computer to go offline. In either case, for a 24 X 7 web site, any time not fully functional is considered a "failure."

The table below offers some real-life examples of what it costs when your internal enterprise systems go offline.

Industry	Application	Cost per Hour
Financial	Brokerage Operations	$6,500,000
Financial	Credit Card Sales	$2,600,000
Media	Pay-per-view	$1,150,000
Retail	Home shopping (TV)	$113,000
Retail	Catalog sales	$90,000
Transportation	Airline Reservations	$89,500

Average Cost per Hour for Internal System Failure by Industry

Note: Data from The Standish Group

For external web sites, from the same Standish Group study, the typical costs were in the middle of the range, running anywhere from $1,000 to $27,000 *per minute* of downtime, or $60,000 to $1,620,000 per hour. Later figures, since 1998, are much higher as companies move more of their business onto the web.

Some examples of external web site failures are:

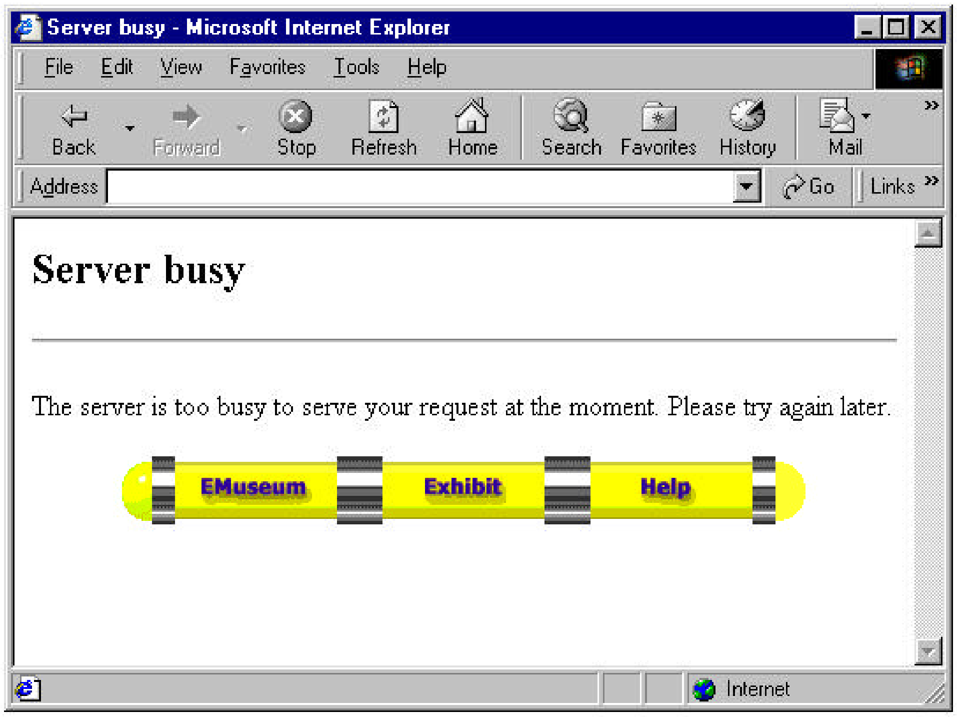
- **ESPN** – Down for 3 days, forced to return $30 fee to nearly 260,000 online "fantasy baseball" players (approximately $7,800,000).
- **Intel** – Earns $275,000 per hour over its web site, a huge potential if a failure should occur.
- **Amazon.Com:** 6-hour downtime: $400,000 in direct costs plus additional in discounts to inconvenienced customers.
- **Egghead.Com:** down for 2 days, estimated $280,000 in lost sales.
- **Charles Schwab & Co.:** After seven prime-time outages in 1999, they added 2 additional mainframes and 66 more web servers (to 154) and are re-architecting their site.

These are the costs associated with a failure *after* the application has been developed and implemented. These types of failures are associated with the stresses and loads that are placed on an application in production usage, and can be minimized or prevented completely with sufficient up-front testing prior to deployment.

Some advantages of performing stress and load testing in addition to functional, usability, platform, integration, and localization testing are:

- Knowledge is strength: the more that the Production Support department knows about the application before it goes into production, the better prepared they are for preventing any unplanned downtime.
- Know how your system will respond: This applies to not only the application being deployed, but also the general corporate data center environment. A poorly implemented application will affect the network, shared servers, backup and restore procedures, and data storage requirements.
- Practice system upgrades safely: Having an efficient test team and their lab equipment available, allows the Production manager the luxury of testing system upgrades (both hardware and software) prior to going live in the data center.
- Find where the bottlenecks are: For an application being developed, stress testing will show areas of the application that need to be re-designed in order to meet the expected user load and performance requirements.

In summary, the goal is to avoid messages like this one:


Server busy - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address Go Links
EMuseum Exhibit Help
Internet

Server busy

The server is too busy to serve your request at the moment. Please try again later.

Reliability vs. Downtime

There are two types of downtime: planned and unplanned. Planned downtime, which has the potential to be just as expensive as unplanned downtime, is typically used for hardware or software upgrades, batch processing, database backup or restore, and system maintenance.

Unplanned downtime is typically caused by hardware or software failures, loss of power to either the computers themselves or the support infrastructure (air conditioning, phone lines, external routers, etc.), or planned downtime, such as a batch job, that doesn't complete on schedule.

Since a 24 X 7 (24 hours a day running 7 days per week) operation is expected to simply always be available, any downtime should be considered a failure. At this point, we need to introduce the concept of reliability as it applies to a 24 X 7 application. Many times, these applications are designed to have "5 nines" reliability, or be available to the customers 99.999% of the time. But in real life, what does this goal translate to?

Reliability	Subsequent Downtime
99%	101 minutes per week
99.9%	11 minutes per week
99.99%	4 minutes *per month*
99.999%	5 minutes *per year*

There are very few operations in the world that can truly meet 5-nines reliability over a long period. However, not all parts of a web site need to meet this goal for the entire web site to be a success. For example, on a consumer site such as Amazon.com®, the shipping department doesn't need to be 24 X 7 since this is an internal function that the customers don't see.

What's a Failure?

One definition for a failure is: *A deviation from the expected delivery or service* (as defined by the International Standards Examination Board, ISEB). Our goal as developers and testers is to anticipate failures and create systems that can survive them. For example, if you anticipate power failures, you can install an uninterruptible power supply, or UPS, and perhaps even a backup generator. However, during one power failure at a manufacturing company in California, the data center had emergency power to their computers, but not to their air conditioning. As the temperature rose through 120 degrees F, the computers started failing. This is an example of not considering all the possible aspects of one type of failure, or of wishful thinking that "it will only be for a short while." One categorization of failures is:

Least Critical

- Productivity losses due to slow performance
- Loss of customers who lose patience with slow site performance
- Temporary inability to access the site. Most will try back later; many won't
- Customers can't place orders once they're on the site
- Computer crashes, customer data is damaged, orders are lost
- Extended downtime, disruption of processes

Most Critical

Failures can also be broken down by internal and external to your company. An internal failure only affects employees, while an external failure involves customers and may involve employees as well. Some of the effects of internal failures are: employees can't get work done, lost time, missed deadlines (and the resultant overtime to make them up), and lowered morale. External failures result in customers not being able to place orders, lost sales, and customer goodwill is damaged – leading to a public relations nightmare. Some failures affect both employees and customers: employees can't produce, your staff is demoralized, production schedules slip resulting in revenue drops, and your insurance costs may go up. If the failure is severe enough, it may result in litigation.

Scheduled Downtime Issues

Applications running 24 X 7 typically have scheduled downtime for system backups, software upgrades, and batch processing. (Translation: they aren't *really* 24 X 7). The results of this offline activity are:

- Productivity impact from closing application files for batch processing
- Business process restrictions due to batch delays, i.e.: your transaction won't be posted until the next business day.
- Users don't have up-to-date information available.
- Impact if batch processing doesn't finish before users start work, or planned downtime turning into unplanned downtime.
- Employees are periodically locked out of the systemWhile batch operations are usually scheduled for after normal work hours (at night), in a true 24 X 7 operation where you have global users accessing your site it's always the middle of the business day somewhere!

Determining Your Cost of Downtime

There are a large number of difficult-to-determine costs that might to be calculated when determining the total cost of the web site going offline. However, many of these can be ignored, as they may not apply to your particular organization.

NOTE: Varying organizations and business models may have additional factors not listed here.

In the example at the end of this section, you'll see that we calculated some of the more obvious direct costs where we could get good figures and then listed, without calculating, many of the in-direct costs.

Cost Factors

Here are some cost factors that should be considered before a failure actually occurs. Many times, if you can show that a "typical" failure would be too expensive, you can use these figures to justify additional resources or schedule in order to avoid the failure in the first place.

Each of these areas will be discussed in more detail below.

- Loss of Productivity
- Damaged Reputation
- Loss of Revenue
- Financial Performance
- Costs to Meet Commitments
- Cost of Repair

Loss of Productivity

Loss of productivity is a very obvious result of your system going down, and also fairly easy to calculate. It's simply: (the number of employees impacted) x (hours out) x (the fully loaded hourly rate for those employees).

Note: The fully loaded hourly rate is that person's salary; benefits; rental (or mortgage) cost of supplying them with an office; infrastructure for the office such as phone; heat; lights; furniture; computer equipment; software; and facilities maintenance. Typically, this is 1.5 to 2.5 times greater than their gross hourly rate.

- How many employees were in the break room waiting for the system to come back up?
- How many employees were working on fixing the problem instead of their current assignments?

Damaged Reputation

While much harder to quantify, your business reputation has a significant impact on your costs and ability to do business. If your "storefront" is consistently unavailable or unreliable, you may find that:

- Customers shop elsewhere
- Less extension of credit by your suppliers
- Lowered credit rating from the financial markets
- Increased credit costs from the banks
- Reduced opportunities (due to your business partners going to your competitors in order to meet *their* commitments)
- Loss of market share

Loss of Revenue

While your system is offline, there is also the loss of revenue from customers not able to use your services or make purchases.

- Direct loss due to customers going elsewhere
- Lost future revenues from customers not revisiting your site once they've settled on your competitor
- Compensatory payments for services not received. For example, in 1999, the ESPN Fantasy Baseball site went offline for 3 days, forcing ESPN to return the $30 fee to nearly 260,000 online players.
- Billing losses (especially if the billing database crashed!)
- Investment losses

Financial Performance

- Revenue recognition will be delayed (best case) or simply lost if customers don't come back.
- Cash flow reduction
- Lost discounts (will effect Accounts Payable)
- Possible short-term loans to meet your debts

- Lowered credit rating (due to missed payments or if your company requires payment restructuring)

Costs to Meet Commitments

Once you've slipped your schedule, you may have to go to extraordinary lengths in order to make up the time and meet your scheduled commitments. This may require:

- Temporary employees, including the time it takes away from your regular employees to hire and train them.
- Equipment rentals to replace any hardware failures, for use by any temporary employees, and to augment your system if it crashed due to a heavy load (such as a seasonal promotion).
- Overtime costs for the extra effort on the part of your regular employees to get the work done while the system is offline.
- Extra shipping costs due to the shorter lead times caused by the slip in your schedule.
- Travel expenses required for executives to meet directly with suppliers and customers in order to reset expectations and keep corporate goodwill.

Cost of Repair

After having a typical application failure, there are the costs of restoring the data integrity and modifying the application in order to prevent future failures.

- Effort required repairing the system after this failure.
- Effort required bringing the system back on line.
- Additional design, development, and test effort in order to prevent this type of failure in the future.

Batch Processing

There are three types of costs associated with batch processing:

1. Cost for locking out users to run batch processing by closing interactive files (only really applies to 24 X 7 operations).
2. Cost for scheduled batch processing in a non-24 X 7 environment that hasn't finished before the users are supposed to start work.
3. Cost for applications and operations people coming in during off hours to do development work on batch programs that they cannot access during normal work hours

1. Batch processing in a 24 X 7 environment

The basic methodology for estimating these costs is to multiply the number of times that the batch job(s) keep people from working per day, times the length of time that people are blocked, times the number of people involved, times their average, fully-loaded wage. For example, in a 24 X 7 operation, if a batch job runs once a night, blocking 30 people from their normal work for an hour and the average fully loaded wage for these employees is $100/hour, we have:

(365 times/year) x (30 people) x (1 hour) x ($100/hour/person) = $1,095,000/year

In other words, if we could design the process to eliminate the nightly batch processing, we could save over 1 million dollars per year!

2. Scheduled batch processing in a non-24 X 7 environment

In a non-24 X 7 operation, any time the scheduled batch processing hasn't finished before people come in to work causes the same type of expense. For example, a nightly batch job runs over into the work day once a week for an hour, blocking 30 people with the same loaded hourly wage of $100/hour results in:

(52 times/year) x (30 people) x (1 hour) x ($100/hour/person) = $156,000 per year.

3. Off-hours development work.

Since the batch jobs block the interactive users from using the system, typically the development of the batch jobs themselves is done on a copy of the application away from the users. However, during implementation and/or repair of the batch jobs, the development and operations team need to use the "live" system to do their work. The cost for this varies by business policy, and can range from zero (where the normally daytime workers are given equivalent time off) to one and a half times the normal salary (or more) if the workers are given overtime. Typically, this is more of a management issue rather than a direct cost.

Armed with numbers such as these, it should be easy to push the analysts and developers towards a non-batch business solution for most applications. The next section will discuss various methods that can be used to minimize the risk of downtime, including eliminating batch processing that locks users out of their application files. After all, if there is no exclusive batch processing, then it can't fail.

Sample Cost of Downtime Calculation

In the pay-per-call support center example below, the plan was for a software upgrade to the telephony switch, along with some system defect fixes, to be put into production usage. Unfortunately, there was insufficient stress testing performed prior to the installation into Production.

The Involved Groups were Information Technology, IT, for the system fixes, the Telephony Group for the switch software upgrades, and Production Support Systems, PSS, for the engineers using the system for customer calls.

The critical part of the process that was missing was the lack of an integration-testing environment that was the equal to the Production environment, resulting in a lack of integration and stress testing at real-life production levels prior to installation.

After the new telephony switch and software fixes were put into place, the system failed under the stress load of normal operations. It took 7 days to completely repair the problem and get business "back to normal."

The result of this 7-day outage was a real and projected loss to the company of $57,884,969. During the computer system outage, the call center engineers used a manual paper system until the automated system could be brought online, and then the manual paper had to be entered into the system after the fact.

Below are the spreadsheet calculations that were performed for this failure:

Cost Factor	Cost	Comment
1. Loss of Productivity		No existing employees were out of work.
Delay to current project work - Un-planned work to IT	$0	Not calculated in this example
Delay to current project work - Un-planned work to Telephony Group	$0	Not calculated in this example
Idle employees	$0	Assumed to be Zero
2. Loss of Revenue		This is money that was not planned for in the current budget.
Direct Loss - Un-planned Dev, Test, and Mgt work to IT	$15,054	IT group was responsible for the development and testing of everything up to the telephone switch.
Direct Loss - Un-planned Dev, Test, and Mgt work to Telephony Group	$6,506	Telephony group was responsible for the development, testing, and installation of the telephone switch software.
Projected - **Lost Future Revenue** - for the Year	$31,204,545	Based on 50% of total lost calls at $500 / call and a 85% non-return rate. Does not include customers not spending as much as usual.
Compensatory payments	$0	Not calculated in this example
Billing losses	$0	Not calculated in this example
Investment losses	$0	Not calculated in this example
3. Cost to meet Commitments		
Un-planned Out Source - **Temporary Employees** - Engineers to PSS	$135,000	Unbudgeted Temporary Engineers were used to assist with call volumes.
Equipment Rental	$0	Not calculated in this example
Overtime Costs	$0	Assumed to be Zero
Extra Shipping Costs	$0	Not calculated in this example
Travel Expenses	$0	Not calculated in this example
4. Damaged Reputation		
Projected Revenue Loss - **Customers (shop elsewhere)** - for the Year	$26,523,864	Based on 85% non-return customers convincing one person not to shop.
Suppliers (less extension of credit)	$0	Not calculated in this example
Financial markets (lowered credit rating)	$0	Not calculated in this example
Banks (increased credit costs)	$0	Not calculated in this example
Business Partners (reduced opportunities)	$0	Not calculated in this example
5. Financial Performance		
Revenue recognition	$0	Not calculated in this example
Cash Flow	$0	Not calculated in this example
Lost discounts (Accounts Payable)	$0	Not calculated in this example
Payment guarantees	$0	Not calculated in this example
Credit rating	$0	Not calculated in this example

Reducing the Risk of Downtime

There are three critical areas for every application that should be looked at for ways to reduce the risk of downtime:

- **Hardware:** Increased redundancy and more powerful equipment with increased excess capacity
- **Software:** Improved application architecture, reduced batch processing, server / network load balancing, and automated monitoring
- **Testing:** In-depth functional testing, load testing, and real-world stress testing on production-equivalent hardware

Each of these areas will be addressed in more detail, below.

Hardware

There are some very straightforward ways to have your hardware provide much needed insurance against your application going offline:

- Build in redundancy for critical hardware components. There are several areas where hardware redundancy can be added, such as using:
    - RAID (redundant array of independent disks) arrays. If a single hard drive fails; there is no data loss. The disk can simply be replaced and the data contained on it will be re-created automatically. If your server is using hot-swappable drives, then the issue is simply to identify the failed disk and replace it without ever taking the server offline. This description is for "RAID 5" which is one of several drive configurations that you might have.
    - Disk mirroring where the contents of the primary disk are replicated onto a second disk. Thus, if the primary drive fails, the system will automatically switch over to the second drive. This is not as safe as using RAID 5, but uses fewer disks to store the same amount of data.
    - A combination of RAID 5 and disk mirroring, or RAID 10. In RAID 10 arrays, the first 5 disks are mirrored on a second set of 5 disks. This allows multiple hard drive failures to occur before the application is impacted. However, the disadvantages of this are slower response times and a large number of disks to store the required data.
    - Storage Area Networks, or SAN's, to store the data. A SAN can be configured as RAID 10 and accessed by multiple computers. Thus it can survive multiple hard drive failures as well as a server failure. The backup server can simply access the same data that the primary server was using without having any lag time between updates to two (or more) separate storage systems.
    - Multiple application servers where the backup server will automatically take over (hot fail-over) should the primary application server fail.
    - Clustered servers for both the web site and database. Then, if any single server needs to be taken offline for upgrading (or a failure), the other servers in the cluster continue processing without interruption.
    - Multiple processor servers allow for the failure of one (or more) CPU's without crashing the application.
    - Multiple sources of power, such as an uninterruptible power supply, UPS, and backup generators as well as the standard power from the utilities company.
    - Multiple geographic locations for your data center's servers. If the primary servers go offline due to fire, flood, or earthquake, having the backup or fail-over servers located offsite will limit the risk of them being affected by the same catastrophe.

- Plan for your application to have at least 50% excess capacity over your peak-anticipated load. Most design work is based on average loads. Normal usage is typically "bunched up" around the start of the business day, over lunch, and again at the end of the day. The average load is not a good indication of the true loads your servers will experience.
- Use hardware load balancing (software load balancing is discussed in the next section) to spread the user load over multiple servers or clusters of servers. Then, should a server begin to respond slowly due to the load on it, additional users would be automatically directed to another server. This type of load balancing also allows the production support technicians to take a server, or cluster of servers, offline for routine maintenance without disrupting the application's availability.

Software

Below are some suggestions for evaluating your software architecture to reduce the risk of application failure affecting your end users:

- Use caching to reduce network traffic.
- Avoid hard-coded network addresses or path names.
- Incorporate batch processing into the interactive data stream to eliminate planned downtime. One method to do this is to perform offline batch processing on a server that will then publish the results via SQL replication to the live OLTP (online transaction processing) server.
- Plan upgradability through configurable hooks into the code and database. For example, in order to avoid having to reboot the database server to accomplish a schema change, add several additional tables to the schema and additional fields to each table, leaving all of them null in the initial implementation. Then, if you need to add fields or tables later after the application has gone live, all you have to do is modify the existing tables and columns to start using the unused tables and columns – and modification doesn't require stopping and starting the database service.
- Design maintainability into the application with modular construction and a high degree of internal documentation in the code. For example, use a COM layer between the UI and the database server in order to isolate the database from UI changes.

Testing

Testing can be broken down into two broad categories: Functional and Advanced (or non-functional).

Functional testing can be described as a combination of the following areas, and is almost always performed to one degree or another on all software projects.

- Functional Testing: does it work?
- Usability Testing: can I do my job with it?
- Integration Testing: does it work with the other applications I use (email, word processing, spreadsheets, etc.)?
- Platform Testing: does it work on my machine (hardware, operating system, browser, network)?
- Localization/Globalization: does it work in several languages, connected together through a common database?

Advanced or non-Functional Testing consists of load, performance, and stress testing and is often skipped due to schedule and resource limitations on the project.

- Load Testing: Measures the design limits of the software - how it behaves with very large files, large data transfer streams, or very large databases.
  - How does it handle large input files?
  - Can you run a report of all the data in the database?
  - Does the server crash when recovering all the data for the past year?
  - At what point will the system hang when manipulating large data files?
- Performance Testing: Measures speed of the application and finds processing bottlenecks. This testing will also provide a measurement for code and architecture changes: did they improve the response time, or make it worse?
  - Does the application responding fast enough for the users?
  - Did this release improve over the previous version?
  - In which module or component is the application spending most of its time? This will determine where the development team should focus their attention in order to improve response time.
  - Does caching help or hinder the users? (Do they have to dump the cache manually to get the latest data?)
- Stress Testing: Allows you to know in advance how many users your application will support and lets you streamline processing during the development stage rather then fixing it later.
  - Can the application handle 150% (or more) of the anticipated number of users that you expect under peak conditions?
  - How many users can it ultimately handle before crashing?
  - Can it handle the interactive load and batch processing at the same time?
  - How does it respond to operating system partial failures? (RAM, disk space, processor)
  - How does it utilize available network bandwidth? (LAN, WAN, dial-up)