# After the Boom: A Health Check for IT Systems

Benedikt Stockebrand
<me@benedikt-stockebrand.de>
http://www.benedikt-stockebrand.de/

March 23, 2003

# Contents

# 1 Introduction

During the last five years the IT world has encountered several revolutionary changes:

- The widely unanticipated end of a whole century caused a flurry of activities, obsoleting tons of decade old hardware and terabytes of software from one second to another. The resulting IT projects covered a huge range of complexity, from the simple replacement of a desktop PC to migrating entire banks from their proprietary COBOL programs to SAP R/3.
- Only two years later the introduction of the Euro as the new currency throughout the European Union caused another similar, although lesser, wave of activity in many areas.
- Alongside these spectacular, and precisely dated, events, the Internet established itself as a new communications medium that forced the entire business world to change their processes.

All together these three events alone made the IT industry boom like never before. Which made a lot of quacks come into the IT world for the quick money. Who left the IT world with a wide range of functionally inadequate, badly designed and badly implemented IT systems.

Now that the excitement is over, things change. The quacks mostly left "to find new challenges" and aren't much of a loss. Managers start to worry about costs again, sometimes to the point that they entirely forget that these costs are what generates enough revenue to make for a proper business plan[1] . But lamenting won't change the situation; instead, existing systems need to be analyzed and then made more profitable—or abandoned if there is no hope of salvaging them.

Now what is a *healthy* system? In the context of this discussion I consider a system "healthy" if it meets requested requirements at reasonable costs. Independent of the particular functionalities involved, requirements can be basically categorized into efficiency, reliability and lifetime expectancy—and in the case of new systems, a reasonably early start of operation.

In a perfect IT world, all these requirements have been specified, analyzed, implemented and successfully and at minimal cost met before a system is first brought into operation. In reality, even obvious functional requirements are often barely met. More hidden requirements, like reliability and extended lifetime expectancy, are usually less obvious to non-technical management and end users and as a consequence even less likely to be met.

Additionally, during the "New Economy" boom, "time to market" considerations often led to quick but expensive solutions. The valid intention to gain "market share" by meeting the "market window" has since been succeeded by cost consciousness and competitiveness. Generally requirements change over time, and sometimes slowly and gradually so the change won't be noticed and properly taken care of.

Finally, IT technology changes fast. What used to be a healthy solution five years ago may be outrageously expensive to maintain and operate today. Or worse, it may be impossible to maintain because some product used has become unavailable.

In this manuscript I present a methodology to analyze the health of an IT system. Following a traditional approach, the requirements would be updated and then compared to the existing system. This approach has two drawbacks: the reluctance of non-technical "business users" to spend some "unnecessary" effort on a system that "already works" and the quickly changing IT technologies already mentioned. So instead I propose a different approach: An analysis what could be done to improve the system and what can and what cannot be expected from an existing system as is, all presented in a form that can be presented to management and business users with the question: "Is this what you want?"

In the scope of this manuscript we won't consider the root causes for the symptoms we observe. Neither will we analyze how to devise a plan how to "cure" a system.

[1] DeMarco [DeM93] has some nice comments on that from a software developer's point of view.

# 2 The System Health Matrix

The analysis follows a two-dimensional schema, assigning the components of the system analyzed to the rows and the analysis criteria to columns.

The rows are furthermore hierarchically structured. There are four major groups: Technical system components, the people involved with the system, functional subsystems and the entire system as a whole.

The columns refer to efficiency during normal operations, risks of system failures, and the lifetime expectancy of the system.

Figure 1 (p. 4) shows a template for such a health matrix.

The rest of this section explains how to fill the rows and columns of the matrix with system subcomponents and detailed evaluation criteria, respectively.

## 2.1 Matrix Rows I: Technical Components

Probably the most obvious candidates for an analysis are the technical system components. To improve chances that an analysis addresses all relevant components it is useful to classify them and work through all the classes.

### 2.1.1 Infrastructure

Even though the infrastructure may be considered "outside" a system, any individual system analysis that is not part of an evaluation of an entire IT environment (like a data center or even a group of data centers) must address the environment in question. After all, problems with the infrastructure may affect the systems health quite seriously.

Infrastructure components that affect the health of an IT system can be:

**Data Center Space:** The space available at a data center can be very critical. Once a system is productive, moving it to another space, possibly another data center, can become an expensive affair, both as far as the actual transport as well as the potential downtime are concerned.

**Power Supply:** Power supplies are important to analyze for two reasons: The external supplies may be unreliable. If they are, the UPS and diesels become more important to analyze both for reliability as well as sizing (of the diesel tanks, in particular). Next, data centers need more and more electricity. Five years ago a rack powered with two 16A lines at 220V was considered well equipped. With the upcoming blade systems it should be obvious that power requirements may easily triple within the next few years.

**UPS and Diesels:** Like the regular power supply, UPS and diesel generators are obviously essential for reliable operations.

**Air Conditioning:** Undersized or unreliable air conditioning affects all systems installed in a data center. At best, a relevant air conditioning failure will require the temporary shutdown of less important systems. At worst, hardware failure and possibly data loss can occur. And since the power used within a single rack keeps increasing, spare capacity and/or the capability to add capacity is essential in the long run.

**Internal Networks:** Even though internal network components, including switches, cabling, patch arrays and such tend to be quite reliable nowadays, they should be considered as far as they have an effect on the system analyzed.

**Internet Uplinks:** Since WAN connections are both expensive as well as notoriously unreliable (compared to LAN connections, that is) they deserve some attention.

This list can, by its very nature, not be complete. But it coversthe most components components I've encountered so far and should generally serve well as a basic checklist to avoid missing the obvious.

| | **Efficiency during normal operations** | **Failure related risks and scenarios** | **Lifetime expectancy** |
|---|---|---|---|
| | Costs/ Operational Effort | Component Failures<br>Security<br>Known Problems...<br>Likely Scenarios...<br>Maximum Desaster | Lifetime Expectancy<br>Upgrade Expenses<br>Scalability<br>Availability<br>New Features...<br>Others... |
| **Technical Components** | | | |
| Infrastructure | | | |
| *Data Center Space* | | | |
| *Power Supply* | | | |
| *UPS and Diesels* | | | |
| *Air Conditioning* | | | |
| *Internal Networks* | | | |
| *Internet Uplinks* | | | |
| ... | | | |
| Hardware | | | |
| *Servers* | | | |
| *Storage* | | | |
| *Clients* | | | |
| ... | | | |
| System Software | | | |
| *Operating System* | | | |
| *Volume Manager* | | | |
| *Cluster Software* | | | |
| ... | | | |
| Middleware | | | |
| *Data Base* | | | |
| *Backup Client* | | | |
| *Monitoring Agent* | | | |
| *Web Server* | | | |
| ... | | | |
| Applications | | | |
| ... | | | |
| **People** | | | |
| Users | | | |
| ... | | | |
| Operations Personnel | | | |
| *Operators* | | | |
| *System Administrators* | | | |
| *DBAs* | | | |
| *External Support* | | | |
| *Help Desk* | | | |
| ... | | | |
| System Developers | | | |
| *Business Users* | | | |
| *Software Architects* | | | |
| *Programmers* | | | |
| *System Architects* | | | |
| **Functional Subsystems** | | | |
| ... | | | |
| **Entire System** | | | |

Figure 1: The IT System Health Matrix

### 2.1.2 Hardware

Next we consider the system hardware, normally the hardware dedicated to the particular system. This usually includes:

**Servers:** With the exception of pure peer-to-peer systems, virtually all systems contain some servers.

**Storage:** As soon as persistent data is stored throughout the system, the storage should be considered separately no matter if it is a single internal disk in a server or a dedicated SAN environment. Doing so tends to simplify the subsequent reliability analysis for the entire system.

**Clients:** Even though dedicated clients are mostly substituted by "multi-purpose" PCs it tends to be useful to take a closer look at the client hardware used.

As with the infrastructure components, this list shouldn't be considered exhaustive.

### 2.1.3 System Software

System software, i.e. software that runs on top of the hardware and provides a hardware-independent interface to the software layers above, is the next class of components we need to address. Common candidates are:

**Operating Systems:** Just the simple fact that a profession "system administrator" exists should demonstrate why operating systems are a vital part of any IT system today and as such deserve particular attention.

**Volume Managers:** In larger IT systems with a significant amount of persistent data, volume managers like VxVM, SDS, HSM, LSM and such are critical components. The server/storage distinction at the hardware level is somewhat mirrored in the OS/volume manager distinction at the system software level.

**Cluster Software:** In theory, using clusters improves a systems availability by providing a "virtual server" running on redundant hardware. But clusters increase the complexity of every system quite noticeably, in some cases to the point that availability is seriously reduced.

### 2.1.4 Middleware

Next are middleware components. They are usually defined as software that provides some application independent high-level functionality to the actual applications running on top of them. The distinction between middleware and application is somewhat fuzzy but serves our purposes reasonably well—unless we intend to start some religious war, of course.

**Data Bases:** Data bases, and RDBMS' in particular, are probably the most common middleware components that have a noticeable effect on the health of IT systems.

**Backup Clients:** Backup clients, while even more common than RDBMS', often go by unnoticed. Until somebody desperately needs to restore some data, that is...

**Monitoring Agents:** Proper monitoring is essential to achieve high reliability and as such monitoring agents are quite similar to backup clients.

**Web Servers:** Web servers are a typical border case between middleware and application software. In a web based system that makes extensive use of CGIs, servlets and such the web server proper may actually be considered a middleware component.

### 2.1.5 Applications

Most IT systems run some application software that provides some functionality to the users. These applications are obviously quite important, but different from the previous categories they can't be pressed into some generic structure. So it takes quite some understanding of the structure of the applications and particular care to consider them properly structured and without missing anything relevant.

## 2.2 Matrix Rows II: People

It is highly fashionable to dismiss employees from consideration by calling them the "companies most valuable asset". Nevertheless we just consider them part of the IT system and assess how critical they are to the health of the system.

According to standard sysadmin lore there are effectively two kinds of people: Sysadmins and the lower life forms. We further distinguish the between three lower life forms: users, system developers and almost-sysadmins.

The three groups of people, users, operations personnel and system developers, correspond to some degree to the three major evaluation criteria of efficiency, reliability and long term prospects—at least they should correspond, provided that a system is in any reasonably healthy state.

### 2.2.1 Users

If an IT system is direcly used by users, it is important to check the interaction between the system and the users.

If the users are working for the company that owns the system, it is obvious that the users "function" as part of the system. It may be possible to obtain some "performance" data from the controlling or human resources department about that interaction. Otherwise, that data needs to be obtained e.g. through interviews and/or some basic calculations based on the number of users and the number of transactions taking place in a given amount of time.

If the users are customers it may be harder to obtain such data. In that case however, it is more likely that marketing, customer care or even top management bother to see how satisfied customers are with the system so they can supply the data.

### 2.2.2 Operations Personnel

Similar to users, operations personnel is quite important to the "functioning" of an IT system. Their role is more extensive, though: They must handle the system during regular operation which in itself can be a tedious and time consuming task. They must also deal with problems quickly and reliably. They must preserve the knowledge about the system, its problems and their solutions and finally they must adjust the system to a large degree to changing circumstances like new operating system versions, security patches and others.

Operations personnel can be to some degree classified according to these groups:

**Operators:** These are the people who take care of routine work, like changing tapes in a tape library, installing machines into racks and such.

**System Administrators:** Sysadmins take care of the system software, network configurations and usually the middleware components except for relational data bases.

**Data Base Administrators:** Since data bases are fairly complex the DBAs are sometimes a separate group from the sysadmins.

**Application Administrators:** Some applications are so complex that they require special application administrators.

**External/Vendor Support Specialists:** External or vendor support specialists are those specialists that you rely upon to solve the very rare or very difficult problems. They range from hardware repair personnel to specialized product gurus.

**Help Desk:** Help desk personnel is the link between the users and the other technical personnel. They can often relay the average users problems in a fairly structured way, which can be quite helpful when analyzing the system.

### 2.2.3 System Developers

Finally, there are the people who were involved creating the system. In many cases, especially during the last five to seven years, many of them were often brought in as hired guns who were rarely available once the initial project ended.

System developers can be roughly classified into these groups:

**Business Users:** They are the people who have an idea what functionality the system should actually provide. They are usually non-technical but have a detailed understanding of the purpose of the system. They are primarily in charge of the requirements management.

**Software Architects:** They design some (application) software to meet the requirements of the business users.

**Programmers:** They implement the software designed by the software architects.

**System Architects:** They design everything except the software to turn the whole thing into a system. "Implementation", or installation, is usually left as an exercise to the operations personnel.

This classification is somewhat simplified and relates to a certain style of project organization which, albeit widely employed, doesn't exactly prove particularly useful in many cases.

## 2.3 Matrix Rows III: Functional Subsystems

With reasonably complex IT systems there is usually a plethora of different functionalities provided. Some of these functionalities are more important or more complex or more reliable than others. It is often helpful to break up the system into such functional subsystems, i.e. subsystems that provide a certain functionality or group of closely related functionalities.

Defining these subsystems is sometimes nontrivial and requires some understanding of the business domain involved. Once the subsystems are defined it is often quite simple to derive the assessment of each subsystem from the assessment of the components and people involved.

Why are the functional subsystems so important? To explain the state of a system to a "technically challenged" management it is mandatory to show the economic relevance of the problems found. The functional subsystems do exactly so; a statement "With the current backup solution it is entirely unknown if it is at all possible to do a desaster recovery in case of a total loss of the disk storage involved, e. g. due to water or fire damage, without manually re-entering all the data from existing paper records" is more useful than a "I've seen some people try to do their backup with tar and gzip on a DAT DDS-2, but I've never seen it work".

## 2.4 Matrix Rows Grand Finale: The Entire System

Finally, all the relevant aspects can be compiled into an overall statement about the system, somewhat like a "management abstract".

Just like other "management abstracts" known elsewhere such a compilation is only useful if the system isn't too complex and the state of its components is somewhat consistent. With more complex system where the components range anywhere from "pristine" to "irrecoverably broken" any oversimplification will be counterproductive.

## 2.5 Matrix Columns: Quality Criteria

Now that we defined *what* we want to analyze it is time to define what criteria we want to apply.

As mentioned in chapter 2 there are three basic criteria relevant to the health of an existing system. Since their natures are quite different we address each of them and explain their peculiarities.

The three criteria are *"efficiency"*, *"reliability"* and *"adaptability"* and they assess the system during normal operation, during abnormal situations and its long term perspective, respectively.

Since we consider already existing systems only, we may safely ignore a fourth criterium that basically applies to systems under construction only: Meeting the launch deadline, or being launched as soon as possible.

## 2.6 Matrix Columns I: Efficiency During Normal Operation

Efficiency is the first order approximation of a systems short term economic health, at least if the system is reasonably reliable. Traditionally, efficiency is measured in "bangs per buck", where "bangs" are measured in whatever unit that meets the functionality provided and "bucks" can be any reasonably hard current.

At first glance, efficiency of technical components seems quite easy to measure; just ask controlling how many bucks you spend and marketing, HR department or management how many bangs you get out of them. Doing so however may reveal that it is difficult to define how to measure the bangs. Taking a look at the functional subsystems may be helpful to define a type of bang or "business transaction" for each of which the matrix should be extended with a separate column.

Similarly, the effort necessary from the people involved during regular operations can be assessed; at worst it may be necessary to measure how long a user needs per bang, assess the effort for operations personnel routine tasks and put that into relation with personnel expenses. System developers should be irrelevant in this context; if they aren't that's an obvious sign of serious problems.

The functional subsystems and the entire system can usually be assessed through basic arithmetics from the already assessed components and shouldn't prove unsolvable to anybody with average arithmetics skills.

So far the efficiency analysis is little more than boring and tedious unless we've been lucky and got a chance to scrounge the necessary information from the controlling department. We've got a few numbers, but by themselves they don't mean anything—we need some others we know to be "reasonable" to compare them to. In an average environment, presenting the numbers found so far to the average technically challenged management will result in some "this is too expensive, we need to cut the costs", which is just the psychologically correct way to say "you, the techno geeks, cut the costs and we, the management, will keep complaining anyway"...

Theoretical computer science shows a way to avoid this situation. The objective of computational complexity is to show *lower bounds* on problems; no matter what (correct) algorithm people come up with, it can't be better than these lower bounds. Any algorithm that approaches the lower bound reasonably well is considered a "good" algorithm.

We can use a similar approach: We dream up an imaginary system that is reasonably well made and derive some "reasonable" reference values from it. We can then compare the values from that imaginary system with the ones from the actual one. If both are reasonably close we can assume that the system is healthy.

The challenge with this approach is that the imaginary system should be unrelated to the existing one—it is meant to be "reasonable" not "like the one we've got". This requires experience, imagination and sometimes some lab tests to obtain useful reference values.

It also requires experience to understand the meaning of "reasonably close" in this context.

## 2.7 Matrix Columns II: Reliability and the Risks of System Failure

In a perfect system the health check should be complete after the efficiency assessment. But in real life, system failures are part of the business. So we need to take reliability and failure issues into consideration to augment the results we got from the efficiency assessment.

While risk analysis and risk management can be made a sophisticated science, for our purposes we limit ourselves to the basic definition that risk is the product of the probability that an event occurs times the damage it will cause. On the damage side we can only assess the immediate technical recovery expenses; all other damages need to be assessed by the business user and/or management. To support them doing so we can provide failure and recovery scenarios and their probability.

There are basically two kinds of failures which I'll call *"non-disruptive"* and *"disruptive"* failures, respectively. While non-disruptive failures, like a disk failure within a RAID subsystem, don't affect the business side, i. e. the users and the money making, disruptive failures do affect users and business. This distinction is somewhat fuzzy; a disk failure in a heavily loaded RAID5 subsystem will cause a performance loss that is noticeable to users while the temporary loss of a primary DNS server and the resulting incapability to do DNS updates will usually go unnoticed in many environments.

This distinction leads to the question if non-disruptive failures shouldn't be considered part of the efficiency column. Ignoring both tax regulations and controlling methodology I rather consider them failures like their disruptive counterparts because even a non-disruptive failure causes unexpected and unscheduled work to the operations personnel. Anybody who had to fix a problem late at night while on call duty should appreciate this choice.

The reliability and risk related columns of the health matrix are more differentiated than the efficiency column. We first consider component failures, where components are not limited to hardware but also include software and the people involved with the system. Next we consider intentional failures due to security related events. Then we make use of experiences made with the system so far and assess problems already known from experience and scenarios we consider likely. Finally we assess an assumed "maximum desaster" and how we deal with it.

### 2.7.1 Component and People Failures

Probably the most obvious type of failure is ahardware failure. We generalize this and consider failure of all technical components as well as people.

With all components it is a matter of experience to estimate how likely a failure is; different from the initial project that created the system we can usually make good use of the experience gathered during the operation of the system.

To assess any invididual component we need to answer three questions:

1. How can the component fail?
2. What are the consequences of each kind of failure?
3. How likely is each kind of failure?

The first question is usually the most difficult to answer once we're past the failure of individual disks within a RAID system. It requires both experience and imagination to come up with a reasonable list of failure types. And it requires knowledge what other components the one considered depends upon in which way.

The second question can be tedious to answer in a complex system; for methodological reasons we won't consider the effects on functional subsystems but focus on the "immediate" consequences. But still, a RDBMS that mysteriously crashed, losing data in the process, can't be revived following some routine procedure. The important point here is to check not only the consequences like "if $A$ breaks then $B$ will lose data and $C$ shuts down automatically" but also the whole sequence

of (potential) events all the way back to normal operations. So it could be something like "if $A$ breaks it takes five minutes to have the monitoring event (or a user complaint) reach sysadmin $B$ who will diagnose the problem in approximately 15 minutes and call vendor $C$ to replace $D$ within another 4 hours according to a service contract and call the backup admin on duty to do a 6 hour recovery, resulting in a loss of service of approximately 10:20 hours and recovery costs of approximately 2000 bucks". (I told you it could be tedious.) To keep the effort at bay as many problems as possible should be considered together; neither the monitoring nor external service contracts need to be documented for each and every such failure. Some common cases along the lines of "nobody has a clue what's going to happpen if $A$ breaks because nobody ever bothered to worry about it[2]", should be documented exactly like this—preferably in red if you have a color printer at hand.

The third question can sometimes be answered from problem tracking systems, log books or otherwise simply from the operations personnels experience.

While it may sound quite mechanistic, people are quite similar to technical components. They age and eventually retire, sometimes they quit their jobs on short notice and in some very unfortunate cases they simply become unavailable because of illnesses or accidents.

People differ from technical components in one fundamental way: They are capable to learn. This provides the system itself with a capability to learn, to adapt and improve itself. Again this sounds quite mechanistic, but the loss of every "human component" that alone knows about some aspect of the system is a partial amnesia of the entire system. Insufficient "knowledge replication" is extremely dangerous because a desaster recovery, i.e. hiring a successor and bringing him/her up to speed, is expensive, takes time and is always only partially successful.

### 2.7.2 Subsystem and Entire System Failure

Assessing subsystems is slightly different from assessing basic technical components but generally follows the same pattern.

The three questions from the previous section remain the same but their answers look a bit different. Subsystems don't fail by themselves but their modes and probabilities of failure can be derived from the components they are built upon and their interaction within the subsystem, providing answers for the first and third question.

The consequences are normally outside the technical dimension and therefore the second question is left to the business user or management to answer. Any technical analysis can only provide failure scenarios and their associated probabilities.

### 2.7.3 Security

While a full security audit of a system is often beyond the scope of a general assessment, critical systems should have been audited before they went online, after major changes and at fixed intervals during their entire lifetime.

So much about theory.

Since a full security audit is out of scope for our purposes we limit ourselves to modify the three questions about failure modes, consequences and probability and ask six questions about the security of components, personnel, functionals subsystem and the entire system:

1. How security-sensitive is each component?
2. What kinds of security related events are possible or imaginable?
3. How likely is each event?
4. How do I notice each event?

[2] Of course, Nobody just left the company for a better job elsewhere...

5. What is the defense plan for each event?

6. What are the consequences of each event?

There are basically two kinds of security events to consider: The discovery of security flaws in system components and actual attacks. Both kinds basically pose questions about the security related quality of the component, its visibility, the notification procedure, vendor support and the associated handling procedure. The second kind, dealing with actual attacks, additionally shows architectural mistakes like insufficiently layered security models, and the business consequences of an attack.

Unanswered questions in this column usually show some serious problem and should be clearly marked as such.

### 2.7.4 Known Problems and Likely Scenarios

Since we are dealing with an existing system where we can make good use of previous experience it also pays to analyze problems that have occured in the past.

This involves interviewing the people involved with the system and, if available, the analysis of log books, problem ticket systems and similar ressources. For each relevant problem found a separate column should be created and then the role of every component with respect to that problem should be assessed.

In this context, "likely" scenarios are those that are considered a reasonable risk. While it may be unlikely that two redundant Internet links simultaneously fail, the consequences and therefore the risk may be high enough to warrant a closer examination.

### 2.7.5 Maximum Desaster

As a special case of a "likely scenario" there should be a "worst case" defined and analyzed. It basically serves as a catch-all for all those problems and scenarios missed because nobody ever thought of them so far.

Dealing with anything beyond this scenario should be entirely out of scope of a technical assessment. The remaining risk is either accepted as inevitable or an adequate insurance should be sought. What to define as the maximum desaster depends on the importance of the system and the available ressources. A bank may possibly consider to deal with the loss of an entire data center due to a major fire by transferring operations to a standby data center. A student writing a master thesis usually won't buy a second, identically equipped PC to deal with the consequences of a strategically spilled cup of coffee.

## 2.8 Matrix Columns III: Adaptability and Long Term Perspectives

So far we have only considered the health of a system at the time of the analysis. While this is quite important, it isn't enough.

A system that requires some special hardware or some particular OS version is bound to be comparatively short-lived. Since virtually all systems once needed an initial economic effort to get started it is important to keep them alive as long as possible with as little economic effort as possible. Unfortunately most initial projects ignore this issue.

What's worse, requirements change. Twenty years ago a bank could still afford to shut down their computers for servicing every night. Today customers expect online banking to be always available. Hardware continuously gets cheaper and faster, so features that would have been economically infeasible five years ago are becoming standard, at least with your fiercest competitor, and need to be added to your system. The Internet branches of many businesses are still growing fast, so systems may keep growing at a rate faster than accordingly faster standard hardware becomes available.

So we need to assess the future prospects of the system we investigate as well as its present state. Together with our primary tools, experience and imagination, this takes some additional level of guesswork. We distinguish roughly between two aspects: The lifetime expectancy of the system with the given functionality and the handling of new requirements and functionalities.

### 2.8.1 Prolonging the Lifetime expectancy

In our matrix we first use a column "expected lifetime" to document what the lifetime expectancy of each component is. With hardware, the important point is how long that hardware will stay supported by its vendor so that spare parts and service contracts are available. Similarly, software components may require certain hardware and vice versa. These dependencies need to be documented here. The "lifetime expectancy" of people should obviously not be taken literally; instead their involvement with the system should be considered, taking int account not only their retirement but a potential frustration related resignation, transfer into another project or the end of their contract. Subsystems are treated like before, by aggretating the data of the components they are built from.

The results found in this column are often even more useful to scare management than an unexpected tax audit.

Next we use another column "upgrade expenses" to document if it is possible and what needs to be done at what costs to extend the lifetime. It is helpful to determine the most critical components first so we know for what time frame we need to extend the lifetime of all other components. This provides some idea of the expenses and effort necessary to keep the system alive and allows management to decide when to plan for a successor system.

If the system is healthy it should have a reasonable lifetime expectancy and acceptable upgrade expenses.

### 2.8.2 Changing Requirements

Finally, the requirements towards an IT system tend to change. For every requirement that is suspected to change another column should be established to document what expenses occur to implement these costs throughout the system.

There are some standard requirements that should be almost universally considered:

**Scalability:** Businesses tend to grow. The Internet side of business tends to grow faster than the rest. So it is important to check for all components and subsystems if they scale.

**Availability:** Availability requirements tend to converge geometrically towards 100%. Designs that don't consider this and e.g. require several hours every night to run an offline backup should have their availability limitations documented.

**New Features:** Often additional features get requested after the system has become productive. If any of those requirements are known or guessable they should be roughly analyzed both for feasibility and costs here.

**Others:** In some cases there are other requirements that can change. These include e.g. changing legal requirements, external interfaces that get updated and such.

# 3 Filling the Matrix

## 3.1 Tools

There is a range of tools available that can be used to create and maintain a matrix.

Probably the most obvious tool is a spreadsheet program; the matrix will get bigger than you can fit on a normal sheet of paper. That doesn't mean that it's a good idea to stuff all

and everything into a single spreadsheet, but keeping a central spreadsheet that refers to separte documents for more complex issues helps to keep things manageable. If you want to show off with the occasional large scale printout you might want to make use of a inkjet plotter if you have access to one.

One of my favourites is asimple time tracking tool, like titrax (also available for Palm PDAs) or gtimer, which allow with little effort to keep track of the time spent on activities (or customers).

Monitoring and trouble ticket systems are often a useful tool if they are already established since they allow to research the history of the system and the problems experienced so far. In smaller environments there is often a traditional log book used by operations personnel.

## 3.2 Practical Considerations

Now that we've defined the rows and columns for our system we've got a huge empty matrix that looks suspiciously like a serious bit of work. This leads to the obvious question: Is the effort to fill the matrix in any way justified?

In a perfect world all relevevant parts of the matrix should have been considered during the initial project and documented in some way that is in essence equivalent to the matrix. In real projects however many of the aspects we find in the matrix have been ignored; in other words, the matrix provides some guidance to identify tasks left unfinished by the project.

In a ISO 9000 style environment it might be considered reasonable to work through the entire matrix. I personally haven't seen any environment where ISO 9000 was actually adhered to throughout everyday work so I can't tell about the use of our matrix in such an environment.

But what about the "average" IT environment that was rushed into existence through a "death march" [You97] or "(pseudo) crunch project" [RR00] without consideration of quality, that was declared a success because nobody wanted to accept responsibility for the money wasted, that subsequently deteriorated because nobody with the necessary competence and authority took care of it and that has been neglected to the point that nobody knows how to handle even common problems since the last sysadmin who knew left the company after a major clash with management?

In such an environment it may be helpful to introduce the matrix to get a general overview of the situation even if the matrix is mostly incomplete. Even more important, continuous updates of the matrix help to understand the impact of problems when they occur. A vendor who goes bancrupt or simply discontinues the support of some products, changes in staff, hardware and software modifications all tend to have hidden dependencies that can be discovered through the matrix. "Continuous updates" in this context doesn't mean that all and every event and action needs to be incorporated into the matrix immediately; updating just before a meeting with management, preferably generating a list of relevant changes since the last meeting along the way, should usually suffice.

Nice about the matrix approach is that it is useful even with a mostly incomplete matrix to identify the impact of problems quite easily, show where minor efforts can at least ease the situation and provide a way to prioritize activities in a sound way. And, last but not least, it helps to communicate with non-technical business users and management, too.

## 3.3 Dealing with Unknowns

Another open question is: How do I deal with missing information?

Consider the archetypical backup issue again: Because some backup has never been tested we don't have any idea how long a desaster recovery will take after some essential storage subsystem, preferably the boot disk, fails. We don't even know how to do that desaster recovery or what problems we have to expect (and deal with) during the recovery.

This is obviously a desaster waiting to happen. But where does it fit into the matrix and how do I recognize similar but less obvious problems?

The simple answer is: In the particular cell ("boot disk","component failure") there should be some explanation what happens then but instead it shows a red question mark. This propagates

to all functional subsystems that depend on the server in question, all the way up to the entire system. Similarly we document the situation with all other components that may need some sort of restore or desaster recovery. All procedures that have never been tested, documented or made general knowledge should be thus marked. The "maximum desaster" column is particularly important here: If it has question marks somewhere throughout the column it should become obvious to everybody that some unexpected desaster is waiting to happen.

## 3.4 Examples

Now that we've seen what the matrix looks like and what should be documented where and how it is time for some war stories to see where they show up in the matrix.

- Last time I ordered some books at a well-known internet book shop I noticed that they replaced their about six stage order procedure by a single page showing all the assumed defaults and buttons to use if you want to change some of their defaults. This saved me maybe two or three minutes of time, them probably some work on their web servers and related backend machines and seemed quite a nice improvement in general.

  Where does this show in our matrix?

  It is an efficiency improvement that affects both the users (customers) and their server hardware even if it might not show up as increased gross income due to other, unrelated events.

- Consider a DNS system consisting of a primary server and say five secondaries. Whenever the primary is down, updates are impossible.

  How do I see if this setup is acceptable?

  This is not a technical but a business decision. What we may know from the matrix is that the primary had an availability of 99.9% in the past, which is equivalent to eight hours of unscheduled downtime per year, and needs an additional two hours of scheduled maintenance downtime per year for disruptive maintenance. The matrix should document how the functional subsystems are affected and with that information we can ask the business user or management if this is acceptable to them or if they prefer a better, but more expensive, solution.

- An low quality text processing software is used throughout a company. The software is known to crash with documents larger than 100 pages approximately every three hours or less, in 10% of all cases ruining the file on disk during that crash in addition to the unsaved changes..

  Where does this show?

  If this problem actually happens, i.e. the software is used for documents of this size, this should be a dedicated column in the "known problems" group. In this column the effort to recreate the lost document and the probability of this event should be documented so that the HR department can assess the resulting non-technical costs.

- Where does the undertrained, overworked sysadmin show?

  Insufficient training shows in various cases: The sysadmin needs external support for reasonably frequent tasks because he/she is unable to do them himself/herself as well as during common failure scenarios.

  Excessive workload is a bit more difficult to find because normally a sysadmin is involved with multiple systems so his/her workload doesn't show in a single matrix. But the known problems and how they were handled in the past may show insufficient availability of sysadmin power. Finally, the workload during regular operations may be excessive, showing that

the sysadmin is doing unnecessary routine tasks that should be automated—a properly configured monitoring system reads log files noticeably faster than a sysadmin.

Generally personnel problems like this should eventually show in the "lifetime expectancy" of the sysadmins involved with the system: if the last three sysadmins resigned after half a year with the system this should be an obvious indicator.

- A system was originally built by a contractor who went out of business later on. What does that imply?

  Throughout the long term perspective columns there should be question marks about upgrades and new features. If that very contractor also provided reasonably frequent last level support, this should show up throughout the reliability columns. Finally, if the help desk was staffed by the contractor and nobody is really comfortable with the system, this should hit the "help desk" row or related.

  A similar situation can be observed if a "99%" complete system is delivered by a contractor and then the developers are all sent to different projects and customers.

- Whenever a security patch needs to be installed it takes about three weeks to have all "decision makers" sign the written request so the sysadmin can actually do the job.

  There should be a column in the "security" column group for this. It should show the impact on the functional systems involved, i.e. "three weeks uncontrollable attack threat per occurence". Together with an estimate of the likelihood of a successful attack and the subsequent damage this should explain the situation even to non-geeks.

# 4 What's Next?

Now that we have a systems health matrix, what do we do next?

Well, that depends on what the intention was to create the matrix. You may decide that the system is technically in such a hopeless shape that it's time to "face new challenges". You may decide that there are plenty opportunities to improve your working life and improve those issues that are within your control, prioritizing them according to the overview you've got from the matrix.

And you may want to confront management with the situation. If you want to do that, just limit yourself to the functional subsystem and entire system row groups. These are what management and/or your business user will understand. At least, from then on nobody can blame you when a desaster strikes that "you never told us". Which doesn't really help anybody, of course.

Maybe management takes over at this point and uses the matrix to improve their risk management, ressources and financial planning.

In many cases the matrix can help to improve communications between technical personnel and management, giving them a chance to change a system in a timely and undisturbing manner to make it economically competitive.

And maybe, in the far future, even the average management will learn to understand its IT systems as obviously quite dynamic things in a highly dynamic environment; that it doesn't pay to rush them into existence with a project just to consider them another "solved problem" that can be forgotten—until the problems increase beyond the pain threshold again, triggering another project.

Well, maybe.

# Epilogue

This is the manuscript of the final talk of the German Unix Users Group (GUUG) "Frühjahrsfachgespräch" (FFG) 2003 held on March 28, 2003 in Bochum, Germany. Special thanks to

Wolfgang Sachs and Jochen Topf for giving me the opportunity to present the ideas from this manuscript on such a prestigious occasion.

I always enjoy suggestions and comments about the subject presented here. The easiest way to contact me is e-mail to `me@benedikt-stockebrand.de`. There are some more ressources available at my home page, `http://www.benedikt-stockebrand.de`, concerning IT reengineering, system architecture and project management.

## References

[DeM93] Tom DeMarco. Why does software cost so much? In *[DeM95]*, chapter 1. Dorset House, 1993.

[DeM95] Tom DeMarco. *Why Does Software Cost So Much?* Dorset House, 1995.

[RR00] Sharon Marsh Roberts and Ken Roberts. Do I want to take this crunch project? In *[WBK00]*, pages 25–42. Dorset House, 2000.

[WBK00] Gerald M. Weinberg, James Bach, and Naomi Karten, editors. *Amplifying Your Effectiveness*. Dorset House, 2000.

[You97] Edward Yourdon. *Death March—The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*. Prentice Hall, 1997.