A

Reliability Calculations and Statistics

In this appendix we will discuss basic probabilistic and statistical coherences and quantities. Our goal is to learn how to use them in real life. The targets of our discussion are hardware components and hardware configurations. Software shows failures of different kinds which do not apply to the findings in this section.

You might say that statistics is about large numbers (of outages) – it does not apply to predicting failures which happen very seldom. Even if this is true, we can use statistics and probabilities to predict reliability when we configure a system. It also helps to identify anomalies during system operations. In particular we will address the following areas:

- In configuring a new system, there are many options. In theory, your vendor of choice should offer the *best* configuration for your needs; in practice many of your hardware vendor's presales consultants fail to do this. And technicians who assemble the systems often do it in a way such that the system works, but does not show the best possible availability. In the end, we want to achieve the best compromise between reliability, cost, and simplicity.
  To overcome this, we can do some basic computations and approximations to understand the consequences of configuration decisions like "what if we used a second power supply?" or "should we configure hotspare disks?"
  In the following, we will discuss the reliability of disk configurations and different redundant array of independent disks (Raid) levels in detail. There are many options to protect our data and the risk of an inadequate configuration is high.
- During the lifetime of your system, problems will occur. Some will be covered by redundancy (like a drive failure in a mirrored configuration), others will have an impact (like a system crash after a CPU failure), up to real downtimes. If we monitor these occurrences, important conclusions can be drawn. We can distinguish between *good*

from *bad* systems and can identify aging systems, which need to be replaced. Computer systems – like all other technology – differ in design quality as well as in manufacturing quality. There are "Monday systems" similar to cars and we need to deal with them in an appropriate way. Such statistics are typically not published: they contain too much political ammunition. Therefore the data of "your" outages is an important information source, and you know that your numbers are true.

A.1 Mathematical Basics

For the following discussion, we use some basic statistical concepts. In this section we repeat the most important formulas – if you are less interested in mathematics, skip this section and look only at the examples given. If you want more details, many undergraduate text books, such as [9], present them.

▸ *Empirical Probability*

If we execute an experiment a times, and get b times a certain event E, then b/a is the empirical probability of E. For large a, we speak of a *probability* for E and can use it for future predictions:

$$p = \frac{b}{a} .$$

▸ *Bernoulli Experiment*

A Bernoulli experiment has exactly two possible outcomes: the probability for event A is p, the probability for the opposite event is $\overline{p} = 1 - p$. If we execute the experiment n times and want to know the probability of getting exactly k times the result E (and $n - k$ times the opposite result $\overline{E}$), the following formula applies:

$$B_{n,p}(k) = p^k (1-p)^{n-k} \binom{n}{k} . \tag{A.1}$$

▸ *Distribution Functions*

We interpret the Bernoulli formula as a distribution function $k \to B_{n,p}(k)$ with the independent variable k. It gives the probabilities that with n experiments the number of results A will be exactly $k = 0, 1, 2, \ldots, n$ times. For large n and small p we can use the following approximations to make real calculations easier. Equation (A.2) is named the *Poisson formula*, and Eq. (A.3) is named the *De Moivre–Laplace formula*;

$$B_{n,p}(k) \approx \frac{(np)^k}{k!} \mathrm{e}^{-np}, \tag{A.2}$$

$$P(k_1 \le x \le k_2) \approx \Phi(x_2) - \Phi(x_1), \tag{A.3}$$

with

$$x_1 = \frac{k_1 - np}{\sqrt{np(1-p)}}, \qquad x_2 = \frac{k_2 - np}{\sqrt{np(1-p)}}.$$

The De Moivre–Laplace formula is used to calculate the probability for an interval of outcomes (instead of a discrete number of outcomes labeled k above). The term $\Phi(x)$ is the *Gaussian function*. It is an integral which has no elementary antiderivative; therefore, it can be calculated only numerically.

▸ *Tests of Hypothesis and Significance*

In reality, we execute only a limited number of experiments, i.e., experience a limited number of system failures during a limited time interval; therefore, the outcomes are not expected to show the given probability. We need to test the *significance* of our result to accept a given probability (e.g., if we got the probability from our system vendor). On the other hand, a given probability can be wrong as well, as its calculation is based on wrong assumptions or without enough experiments to test it.

In order to evaluate the significance of a result, a so-called hypothesis test is done. We first assume that the probability $p(A)$ for result A is correct and gives the distribution $k \to B_{n,p}(k)$. We make n experiments to test the hypothesis. We expect np times the result A, that is where $B_{n,p}(k)$ has its maximum. If the measured result deviates very much from np, then we might need to disapprove the hypothesis – the assumed probability p might be wrong. In other words, we talk about the probability that a measured result follows a given probability.

In practice this is done as follows. We calculate the sum $B_{n,p}(0) + B_{n,p}(1) + \cdots + B_{n,p}(i)$ for increasing i until the sum exceeds a given limit. Depending on the real numbers, the De Moivre–Laplace formula can be helpful. This limit describes the requested quality of our result. Typical values for this sum limit are 0.975 or 0.995. This belongs to a level of significance of 0.05 or 0.01, respectively. Using this method, we know that the probability of getting $k \ge i$ is very small (according to our earlier definition). We reject the hypothesis, when the measured result belongs to the interval $[n-i, n]$.

▸ *Confidence Interval for Probabilities*

This is a similar circumstance but from a different viewpoint: we made n experiments and experienced f failures. How close is our measured failure probability f/n to the real probability p? This question can obviously not be answered, but we can calculate an *interval of probabilities* $[p_1, p_2]$

Table A.1. Confidence levels γ and corresponding values of c

γ (%)	c
80	1.28
90	1.65
95	1.96
98	2.33
99	2.58

which contains the real probability p with a chosen confidence level γ. If we set γ very close to 1, this interval becomes very large. It depends on the number of experiments and the value chosen for γ to achieve a small enough (i.e., meaningful) interval of probabilities.

The following formula is the criterion to calculate the interval $[p_1, p_2]$. It denotes the probability Q for that interval with confidence γ:

$$Q\left(\left|\frac{f}{n} - p\right| \leq c\sqrt{\frac{p(1-p)}{n}}\right) \approx \gamma = 2\Phi(c) - 1\,. \tag{A.4}$$

The value of c is calculated from γ, using the Gaussian function. Table A.1 gives the values of c for typical values of γ.

If we have done our experiments, measured f and n, and have chosen the confidence level γ, we use Table A.1 to get c. Then we can calculate the interval of probability $[p_1, p_2]$ by solving the following quadratic equation:

$$\left(\frac{f}{n} - p_{1,2}\right)^2 = c^2 \frac{p_{1,2}(1-p_{1,2})}{n}\,. \tag{A.5}$$

This result can be interpreted as follows. If we would do many experiments, a fraction γ of the measured probabilities $\frac{f_i}{n_i}$ would be located inside the interval $[p_1, p_2]$.

A.2 Mean Time Between Failures and Annual Failure Rate

The reliability of a component (e.g., a disk drive, or a controller card) or of a whole system is measured by the *mean time between failures* (MTBF). It is the average time until a failure happens and is typically provided in hours (to make it look more authoritative) or better in years. A MTBF of 10 years means that, on average, every 10 years a failure occurs, based on a large sample. These numbers are provided in the component data sheets of the hardware manufacturers; sometimes they are also provided for whole systems.

There are two problems associated with using the MTBF:

1. The values can vary significantly, especially for components with moving parts, like disk drives. This is because the MTBF depends greatly on the quality of the individual production batch, which is far from constant. Manufacturers obviously try to detect quality problems, but this is not easy, as the manufacturing of complex components is more an art than a science. There are too many variables, like tuning of the machinery, impurities in the clean room, quality of the materials, etc. And then there is also the soft-factor microcode...
2. The appearance of failures does not follow a uniform distribution. The failure rate is high for new equipment (*early mortality*) and if equipment reaches its end of life. The time in-between is when we want to use the equipment for production. We will discuss this *bathtub curve* in Sect. A.6.

However, manufacturers often provide impressive numbers for MTBF, 10^6-h run time (about 114 years) for a disk drive is a standard value nowadays.

The inverse of the MTBF is the failure rate. The *annual failure rate* (AFR) is defined as the average number of failures per year:

$$\mathrm{AFR} = \frac{1}{\mathrm{MTBF}_{\mathrm{years}}} = \frac{8760}{\mathrm{MTBF}_{\mathrm{hours}}} .$$

The AFR is a relative frequency of occurrence – it can be interpreted as a probability $p(A)$ if AFR < 1, where $p(A)$ means the probability that the component (or system) A fails in one year. If you multiply the AFR with the time interval you consider, you get the expected number of failures in this time interval.

For example, for a disk drive with an MTBF of 34 years, the corresponding AFR is 0.029 failures per year. If your disk subsystem contains 200 such drives, you can expect a failure every 2 months.

Even if such numbers can be discussed at length, we provide examples from our own experience based on a sample of several thousand components and hundreds of systems. These numbers should give you a rough idea of the order of magnitude and we will provide "real" examples later on. We see that all the numbers are in a similar range.

A.3 Redundancy and Probability of Failures

We consider two ways to combine components in a system or subsystem: they can back up each other, (e.g., redundant network cards, or mirrored disks), or both are needed for function of the combined components (e.g., a cable connected to a network card, or a stripe of two disks). The first

Table A.2. Typical mean time between failures (*MTBF*) and annual failure rate (*AFR*) values

Component	MTBF (h)	MTBF (years)	AFR (failures per year)
Disk drive	300 000	34	0.0292
Power supply	150 000	17	0.0584
Fan	250 000	28	0.0350
Interface card	200 000	23	0.0438

combination is obviously more reliable than its components; the second is less reliable than its weakest component.

In principle we could use such blocks to build arbitrary complex systems, and the following calculation could cover this case. However, we now concentrate on typical disk configurations and discuss first disk mirrors and stripes. This is the most important application and can easily be done in practice. More complex combinations are typically analyzed by special simulation software.

Let us start with two simple examples which set the basis for all further calculations. Let $p(A)$ be the probability that part A fails, and $p(\overline{A})$ the complementary probability that part A does not fail, $p(\overline{A}) = 1 - p(A)$. This probability is given in relation to a time interval. Let us now consider an n-way mirror. It will fail only, if all disks fail. We use the binomial distribution to calculate the probability of this failure:

$$p_{\text{mirror}} = \binom{n}{n} p(A)^n \, (1 - p(A))^0 = p(A)^n \,, \tag{A.6}$$

where $\binom{n}{n}$ is the number of possible combinations of failure.

On the other hand, a stripe of n disks will fail, when 1, 2, ..., or all n disks fail, or, the system will only work if all disks are OK:

$$p_{\text{all disks work}} = \binom{n}{n} (1 - p(A))^n p(A)^0 = (1 - p(A))^n \,.$$

That gives the failure probability that at least one disk will fail as

$$p_{\text{stripe}} = 1 - (1 - p(A))^n \,. \tag{A.7}$$

Let us go back to the two-fold redundant systems: those are systems which survive even if one disk fails, but fail if a second disk fails (in some configurations, only specific combinations of failing disks lead to a system failure). Specifically, the system only fails, if the *second* redundant disk fails during the outage of the first failed disk. This circumstance is not

reflected in the general Eqs. (A.6) and (A.7). We need to consider the time intervals in question. Here a new term, the *mean time to repair* (MTTR), comes up. We divide a year (8760 h) by the time that is needed to repair the failed part, to get 8760/MTTR possible intervals of repair actions. The probability for failure during such a repair activity $p(A)$ is given by

$$p(A) = \frac{\text{AFR}}{8760/\text{MTTR}} = R\,, \tag{A.8}$$

where R is the *repair time failure rate*. If we look for failure during a repair interval, we have to substitute $p(A)$ in Eqs. (A.6) and (A.7) with R. Because these probabilities relate to only one interval, we have to multiply the end result with 8760/MTTR to get the average failure rate of the whole system in 1 year.

With these formulas we could calculate arbitrary configurations; however, for more complex configurations computer-modeling software is used in practice.

A.4 Raid Configurations

Section 5.2.1 on p. 109 explained how Raid, the redundancy method for disks, works. Let us now calculate the AFRs for different Raid levels: first we compare Raid10 with Raid01. Raid10 is a stripe of mirrored disks with the failure probability of Eq. (A.6); Raid01 is a mirror of two stripes, with a failure probability of Eq. (A.7). These configurations were already illustrated in Figs. 5.8 and 5.9.

The main difference is the situation after one disk has failed: in a Raid10 configuration, we now have a single point of failure which is one disk (the mirror to the failed disk); with Raid01, we have multiple single points of failure, any disk in the not failed stripe would lead to a failure of the whole system.

▸ *Raid01*

With Eqs. (A.6) and (A.7), we get for the failure rate of a Raid01 system with $2n$ disks

$$\text{AFR}_{\text{Raid01}} = \left(1-(1-R)^n\right)^2 \frac{8760}{\text{MTTR}}\,. \tag{A.9}$$

▸ *Raid10*

Using the same approach, we get

$$\text{AFR}_{\text{Raid10}} = \left(1-\left(1-R^2\right)^n\right) \frac{8760}{\text{MTTR}}\,. \tag{A.10}$$

Let us consider $\mathrm{AFR}_{\mathrm{Raid10}}$ and $\mathrm{AFR}_{\mathrm{Raid01}}$ using the Bernoulli approximation $(1-x)^n \approx 1-nx$ for small x:

$$\begin{aligned} \mathrm{AFR}_{\mathrm{Raid01}} &\approx n^2 R^2 \frac{8760}{\mathrm{MTTR}}, \\ \mathrm{AFR}_{\mathrm{Raid10}} &\approx nR^2 \frac{8760}{\mathrm{MTTR}}. \end{aligned}$$

That shows that the failure rate of Raid01 increases quadratically with increasing number of disks n, whereas for Raid10 it grows only linearly.

▸ *Raid5 and Raid3*

Let us now consider a system with "normal" Raid protection. This means a system with n data disks and one additional disk taking parity information. It applies to both Raid3 and Raid5 protection: a failure occurs, if at least two disks fail simultaneously (which means the second disk fails before the repair activity of the first disk is finished). We will label both cases with *Raid5*.

Here the binomial distribution gives the solution

$$\begin{aligned} \mathrm{AFR}_{\mathrm{Raid5}} &= \left[1-\left(\binom{n+1}{0}R^0(1-R)^{n+1} \right.\right. \\ &\quad \left.\left. +\binom{n+1}{1}R^1(1-R)^n\right)\right]\frac{8760}{\mathrm{MTTR}} \\ &= \left[1-((1-R)^{n+1}+(n+1)R(1-R)^n)\right]\frac{8760}{\mathrm{MTTR}}. \quad \text{(A.11)} \end{aligned}$$

We simplify this equation using the Bernoulli equation to second order to achieve a formula which can be used easily in practice:

$$\mathrm{AFR}_{\mathrm{Raid5}} \approx \frac{n(n+1)}{2}R^2.$$

Note that the failure rate increases faster than n^2 with increasing number of disks n. To calculate $\mathrm{MTBF}_{\mathrm{Raid5}}$ (which is the mean time to a real data loss, sometimes labeled MTTDL), we have to divide the hours of 1 year by $\mathrm{AFR}_{\mathrm{Raid5}}$:

$$\mathrm{MTBF}_{\mathrm{Raid5}} = \frac{2\times \mathrm{MTBF}_{\mathrm{disk}}^2}{n(n+1)\mathrm{MTTR}}.$$

▸ *Double Parity Raid*

Several systems introduced recently a higher protection, *double parity Raid* (also called Raid6 or RaidDP). That uses a second disk for data protection, which holds additional parity information. The system only fails,

if three (or more) disks fail simultaneously. To calculate the probability, we use the same approach as before:

$$\begin{aligned}
\mathrm{AFR}_{\mathrm{RaidDP}} &= \left[1-\left(\binom{n+2}{0}R^0(1-R)^{n+2}\right.\right. \\
&\quad +\binom{n+2}{1}R^1(1-R)^{n+1} \\
&\quad \left.\left.+\binom{n+2}{2}R^2(1-R)^n\right)\right]\frac{8760}{\mathrm{MTTR}} \\
&= \left[1-(1-R)^n\right. \\
&\quad \left.\times\left(1+nR+\frac{n}{2}R^2+\frac{n^2}{2}R^2\right)\right]\frac{8760}{\mathrm{MTTR}}\,. \qquad \text{(A.12)}
\end{aligned}$$

From a mathematical viewpoint many more configurations are possible, and these use more disks for parity information and can recover more simultaneous single-disk failures [14]. However, none of them are implemented in products today. With current technologies we see no need to go beyond RaidDP. The additional complexity of even higher Raid levels, as well as negative performance implications (i.e., to calculate the parity information and to read/write to the additional disks) does not pay off.

Comparison of Raid Configurations

Let us now apply Eqs. (A.9)–(A.11) and (A.12) to understand the reliability of different disk configurations, and get a feeling for the numbers.

Let us first look to the dependency of $\mathrm{AFR}_{\mathrm{system}}$ on different numbers of data disks (Fig. A.1).

"Number of data disks" means that we do not count the disks which provide redundancy. In this view the same number of disks also means the same usable data capacity. It is interesting to use it, when we want to discuss how much capacity a Raid group could or should have. For one disk, Raid01, Raid10, and Raid5 give the same number, as in this case we have one disk and one mirror disk for all Raid levels. One data disk for RaidDP means that we have a total of three disks, one for data and two additional disks for redundancy. Obviously, this leads to a much better (smaller) AFR. With increasing number of disks, the AFR increases with different slopes. Raid01 shows the worst behavior, as expected. Raid10 is better than Raid5, because with Raid5, two arbitrary disk failures will lead to data loss. With Raid10, after one disk has failed there is only one other disk which would lead to data loss if it failed.

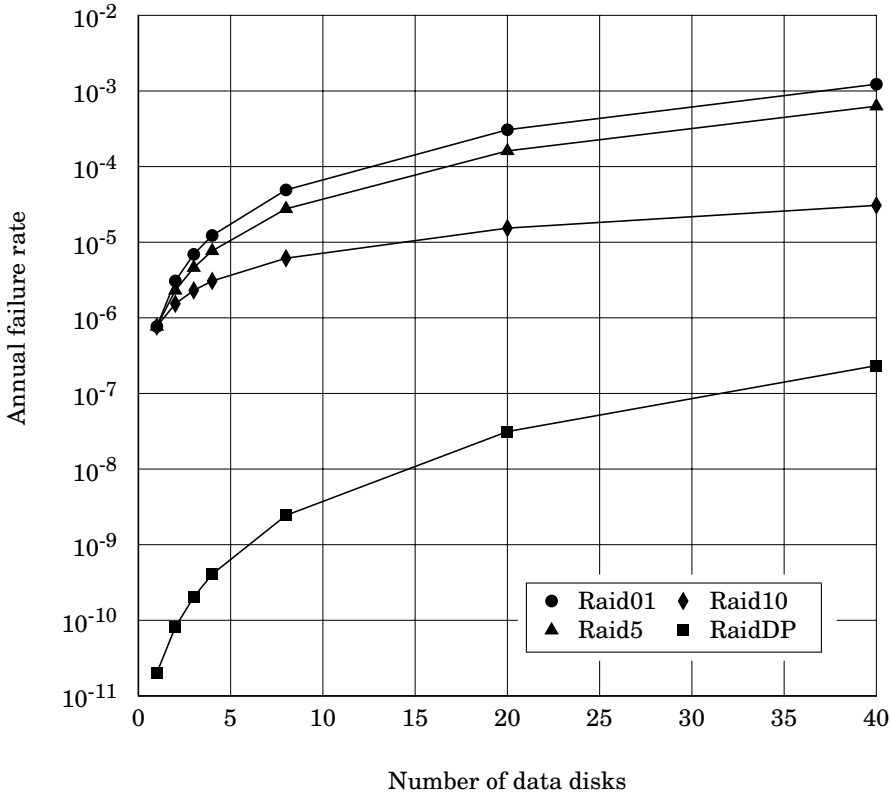

Fig. A.1. Dependence of the annual failure rate for different disk configurations on the number of data disks (see text). The annual failure rate for a single disk is 0.029; the time to repair of a single disk is 8 h (assuming a hot-spare disk – 8 h is the time for synchronizing the data)

It is not a well-known fact that Raid10 shows better protection than Raid5, and is about a factor of 10 for 20 data disks. This is an additional advantage of Raid10 besides the better performance. However, this advantage comes at the additional cost of more disks needed for redundancy, as we will see later.

RaidDP shows by far the best values, which is expected as here three disks need to fail to get data loss. But with higher numbers of data disks (some hundreds), the lines for Raid10 and RaidDP cross – then Raid10 becomes the most reliable configuration. This is when the probability that three arbitrary disks will fail becomes higher than the that for the failure of a specific pair of disks.

In the configuration considered we assume the existence of a hot-spare disk, which translates to a short time to repair a failed disk of 8 h.[1] Without a hot spare, a repair time of 48 h is realistic. This would lead to 6 times higher AFR values for Raid01, Raid10, and Raid5, and a 36 times higher value for RaidDP.

But a hot spare is not only good for improving the AFR values. It also allows the real repair action (when the technician replaces the broken disk drive) to take place on a (again) redundant configuration. This gives protection against human error, e.g., when the technician replaces the wrong disk, which happens from time to time.

Let us now review how many disks a Raid group should have. For this discussion we plot the same instances but in relation of the total number of disks, see Fig. A.2 on the following page. Here Raid5 is worst (but we need to note that Raid5 provides most disk space). RaidDP is still the best, as we expected.

Both graphs from Figs. A.1 and A.2 look at the situation from one perspective, but do not show the full picture. Let us combine the number of disks and the capacity provided in Fig. A.3 on p. 371. Assume that we want to combine eight disks into one raid group. What are the possible capacities and corresponding AFRs? Raid5 shows the most capacity, but the worst AFR. RaidDP seems to be the best compromise between (very good) AFR and capacity. The other Raid levels are not competitive in this scenario (which ignores features like performance).

If we look at Figs. A.1–A.3, all the AFR numbers look very good – why should we care at all? First, these are *averages*: in order to have a good level of confidence for the real situation, the numbers need to be much better than the average. Second, there are circumstances when multiple disks are unavailable at the same time (e.g., controller failure) – this will be discussed later. Third, there are bad batches of disks, which can show significantly higher $\mathrm{AFR}_{\mathrm{disk}}$ values for the single disk than the number provided by the vendor. And, to make this worse, if you got disks from a bad batch, you can expect that many disks from that batch are in your array. How bad can $\mathrm{AFR}_{\mathrm{disk}}$ be? That is hard to say, as such numbers are typically not measured or are not published. Our experience is that a factor of 20 is realistic. Figure A.4 on p. 372 is the same as Fig. A.1, but with 20 times higher $\mathrm{AFR}_{\mathrm{disk}}$. These numbers speak for themselves – we need to be careful!

[1] Note that the time for data synchronization to the hot-spare disk depends on the Raid level: Raid10 is fastest, here only one disk needs to be copied over to the hot-spare disk; Raid5 and RaidDP need more time, as here all disks need to be read in order to calculate the parity information. This effect will not change the general picture we describe here.

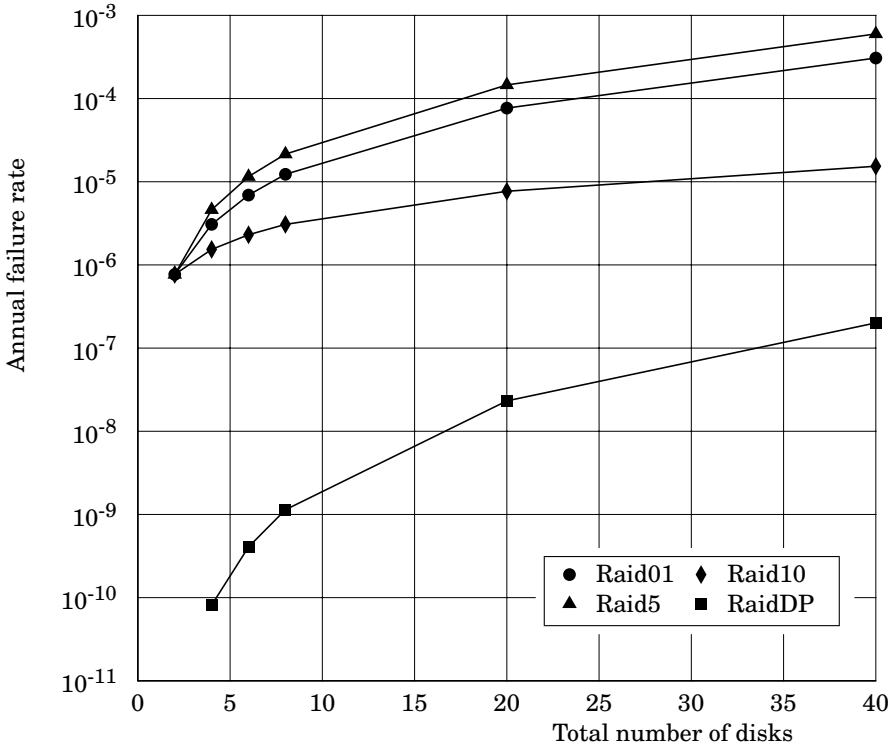

Fig. A.2. Dependence of the annual failure rate for different disk configurations on the total number of disks in a redundant array of independent disks (*Raid*) group (see text). The annual failure rate for a single disk is 0.029; the time to repair of a single disk is 8 h (assuming a hot-spare disk – 8 h is the time for synchronizing the data)

So far we have only looked at disk failures. But what happens if several disks are unavailable at the same time, caused by failure of an underlying component? Let us consider a fictitious disk array which consist of two frames with eight disk drives each. Both are independently connected to a redundant controller. Even if each frame is supposed to be redundant in itself, it can fail. Such a failure can be caused by a microcode issue, or if one disk fails in a way such that it blocks all other disks in that frame.[2] It is obviously a good idea to have the disks and their mirrors in different frames.

[2] This can happen, if the disks in a frame are connected with a Fibre Channel Arbitrated Loop (FCAL).

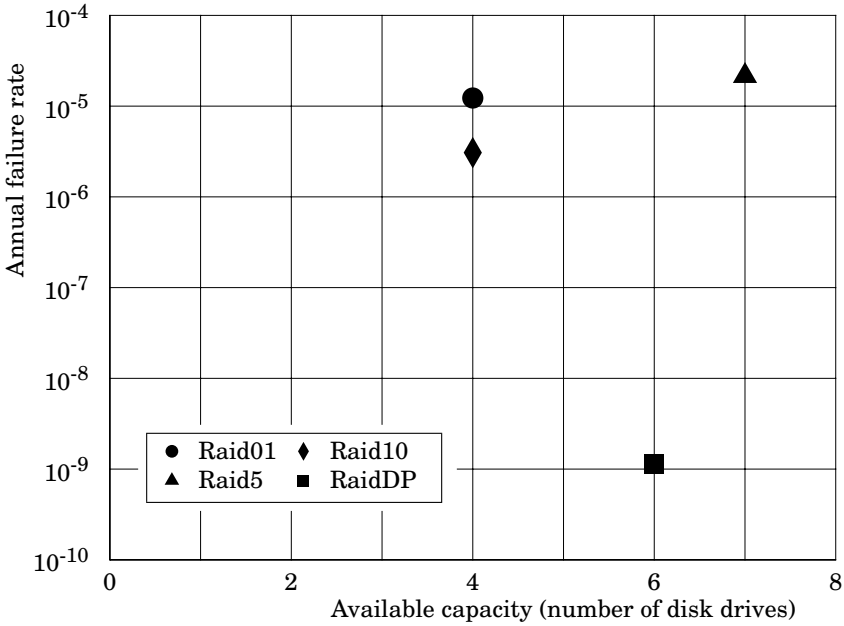

Fig. A.3. Annual failure rate for different disk configurations, all using a total of eight disk drives. The annual failure fate for a single disk is 0.029; the time to repair of a single disk is 8 h

Guidelines

At the end of this section, we give some general guidelines:

- Never use Raid01, unless you use only $2+2$ disks.
- Always use hot-spare disks.
- The AFR for a Raid group should be smaller than 10^{-4}. This maximizes the number of disks to 8 $(7+1)$ for Raid3 and Raid5, 16 $(8+8)$ for Raid10, and 20 $(18+2)$ for RaidDP. For more disks in one Raid group, a closer review is proposed.
- RaidDP provides the highest reliability. It should be used, if performance and hot-spare synchronization time are sufficient.
- Make a full failure mode analysis of your disk subsystem. If there are multiple failure scenarios, they all contribute to $\mathrm{AFR}_{\mathrm{system}}$. There is no template available for how to do such an analysis – you need to understand the architecture and the implementation on a high level and perhaps do some testing.

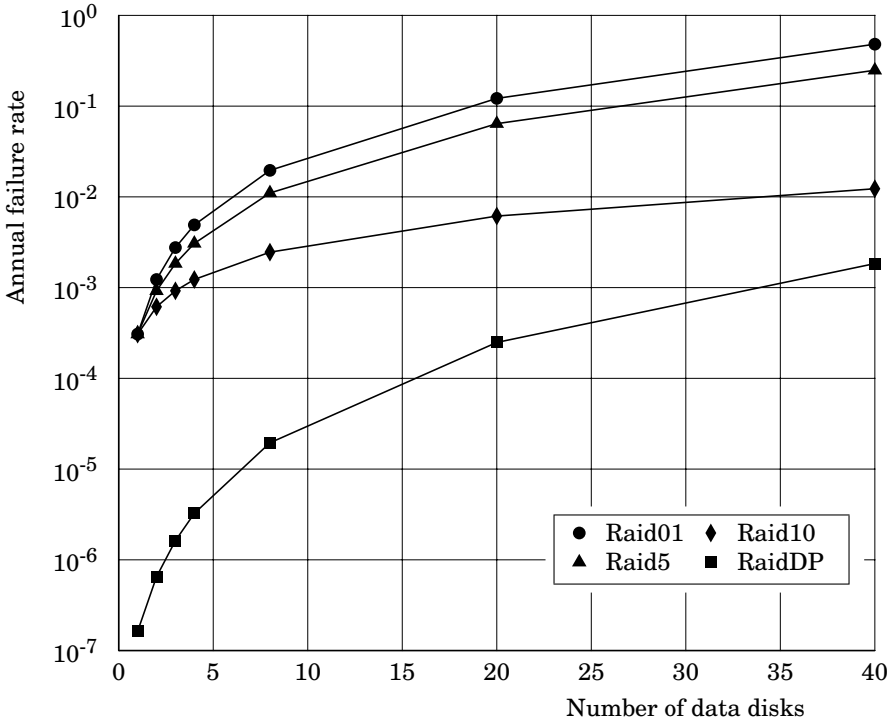

Fig. A.4. Dependence of the annual failure rate for different disk configurations on the number of data disks (see text). The annual failure rate for a single disk is 0.58 – a value for a bad charge; the time to repair of a single disk is 8 h (assuming a hot-spare disk – 8 h is the time for synchronizing the data)

A.5 Example Calculations

In this section we investigate the probability of deviations from the calculated averages. For example, assume a specific value for the AFR: How large is the probability to have one, two, or three failures in one year?

We calculate this using the Poisson formula (Eq. A.2). We use n as the time interval and $p = \text{AFR}$.

Example 1. Let AFR = 0.0167. That means the device is expected to fail 167 times in 10 000 years, or 0.05 times in 3 years (which we assume as the lifetime of the system in question). With the given probability of 5% we do not expect the device to fail at all. But on the other hand, it could fail once, twice, or even more often. The Poisson distribution allows us to calculate probabilities for that; the result is shown in Table A.3 on the next page.

The calculation shows that the probability of getting one or more failures in 3 years is 4.9%, a possible situation. But if we experience two or

Table A.3. Probability for different number of failures.

Number of failures	Probability
0	0.951
1	0.048
2	0.0011
3	0.000019
> 0	0.049

more failures, we would disbelieve the value given for the AFR. This is according to the test method we have already described. ■

Example 2. We have a new – low-cost – computer cluster that consist of 512 nodes (i.e., 512 computers which perform numerical calculations). In the first month it experienced 24 failures, which corresponds to an AFR of 0.56 per node. We want to do a forecast of our expected maintenance effort and therefore we want to know the expected variation with a confidence of 95%. We solve Eq. (A.5) with $f = 24$, $n = 512$, and $\gamma = 0.95$. Then we use Table A.1 on p. 362 to get $c = 1.96$. With that we can determine the interval $[p_1, p_2] = [0.035, 0.061]$. This means that we can expect between 18 and 31 failures per month. We decide to use 30 failures per month for our cost calculation, as some kind of worst-case scenario. It is important to check the evolution of the failure rates every month, to identify possible trends (see Sect. A.6); therefore we repeat this calculation each month. But in this first month, this is the best we can do. ■

Example 3. We purchased a new storage system which contains 200 disk drives. Our vendor provided us with an AFR value of 0.0167 for each disk of the new system. We use the method of hypothesis tests to decide if the new system confirms the AFR value given. We want to be 95% sure about our assessment. We calculate how many disks are allowed to fail in 1 year so that we can believe the value, using a table with the following columns (refer to Eq. A.1):

$$
\begin{aligned}
k &= \text{number of failed disks}, \\
B_{200,0.0167}(k) &= \text{probability for exactly } k \text{ disks to fail}, \\
\sum_{i=0}^{k} B_{200,0.0167}(i) &= \text{probability for a maximum of } k \text{ disks to fail.} \\
&\quad \text{This needs to be compared with our} \\
&\quad \text{confidence level, } 95\% \ .
\end{aligned}
$$

Table A.4 on the next page shows the result: if eight or more disks fail, we can be 97.6% sure that the AFR value given was to small.

In this example we ignore the possibility that the given AFR value could be too large – as customers we are not interested in that case. ■

Table A.4. Verification of a given AFR value

k	$B(k)$ (%)	$\sum_{i=0}^{k} B(i)$ (%)
0	3.5	
1	11.8	15.3
2	19.7	35.0
3	22.0	57.0
4	18.4	75.4
5	12.3	87.7
6	6.7	94.4
7	3.2	97.6

A.6 Reliability over Time – the Bathtub Curve

So far, we have assumed that the probability for failures is homogeneously distributed – independent of the age of a system. This is not correct, as parts and whole systems show more failures at the beginning of their life ("early mortality"), when the so-called *burn in* takes place: Weak systems which were produced outside manufacturing tolerances fail very early. At the "end of life," systems show an increasing failure rate because their mechanical parts or electronics wear out. The probability follows a so-called *bathtub curve* as shown in Fig. A.5 on the facing page. The stable failure period is the time interval when we want to use a system in production. During this interval the weak systems have been weeded out and failures happen in a random fashion with a homogeneous distribution, as we assumed in the calculations before. It is a challenge to identify the time when our system in question approaches and leaves this "good" interval.

There is a combination of reasons for this bathtub behavior. Hardware shows early mortality because of production issues like not meeting required tolerances. Good vendors invest to identify early mortality by running the systems under stress conditions before they are delivered to customers. The "burn in" is done by running them in climate exposure test cabinets, where they run under low and high temperature, and high humidity. They run under vibration conditions to identify mechanical issues like bad contacts or bad solder joints, and also high and low voltage. The goal is to walk "down" the bathtub to the middle part before systems are delivered to clients. Such good vendors are proud of their efforts to not send early mortality systems to customers. Ask your vendor about this to find out what tests are done and if they are done for *all* systems delivered to clients, not only for some control samples.

It is hard to predict when a system is at the end of its life. A rule of thumb is that after 5 years of continuous operations, a system is expected to show more failures.

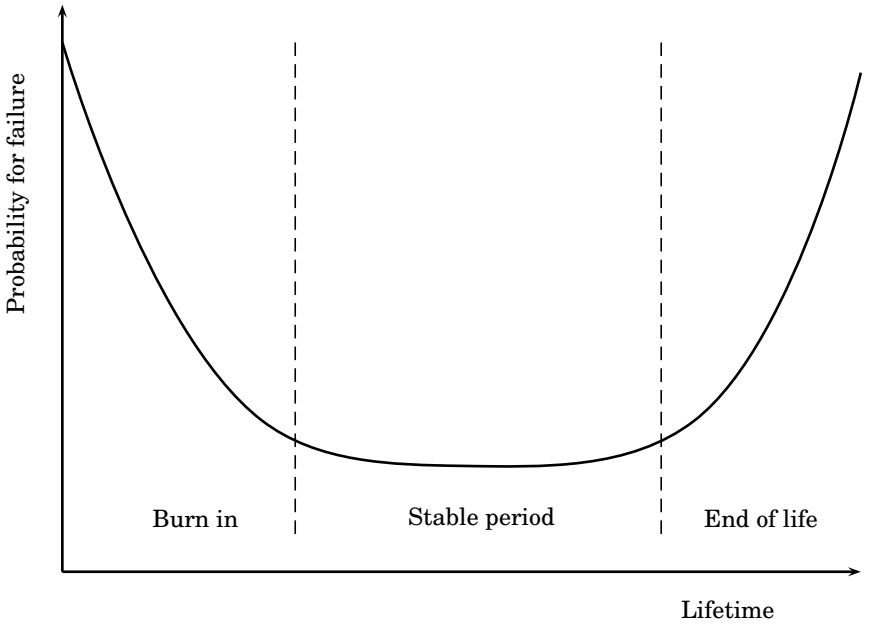

Fig. A.5. Typical bathtub curve of a system

But there is an additional reason for the bathtub behavior – perhaps equally important: the lifetime of a system is similar to or even longer than todays product cycles. If we purchase a new system, we typically want to get the best and greatest: the newest model which was just announced. The risk here is that the new, much faster and better system is not mature yet. If we purchase an early system, the vendor has no experience with it. Microcode might not be optimized for it and integration of the different components might not have been worked out well. Vendors constantly optimize their systems, based on experience in the field. Sometimes a system purchased 1 year after its introduction looks quite different from the early systems. We need to find the right compromise between product innovation and maturity.

It is hard to provide good guidance for how to deal with this situation. If a system is brand new, and the vendor announces a breakthrough from the old one, caution is required. A factor of 10 for the AFR of the newest system and the same model after 1 year of production can be expected. If we purchase such a system, we need to review the company's track record and, if possible, our own experience with that vendor. We should also review what is really new in that system design. Does it show new technologies, or just improve on existing ones? A personal relation-

ship with the sales representative is also helpful, as he or she often has insight into the potential difficulties before and on announcement of the new system. If our system runs until the end of its product life cycle, we have also a negative effect: the system is now expected to be very mature, but it is not more in the mainstream. This means that integration tests are focused on systems from the current (newer) products; patches and microcode changes are first implemented on the new series and so on. A system from an old series in a new environment becomes more and more a contaminant, which is a reason for problems and failures.

The problem of nonconstant failure rates was tackled by the Swedish researcher *Waloddi Weibull* (1887–1979). He defined a distribution function which is widely used in the area of quality engineering which now bears his name [11].

The combination of all these effects leads to the bathtub curve. We should be conservative and stay in the mainstream: our systems should not be too young and not too old.

B

Data Centers

Data centers are facilities (rooms or complete buildings) where IT systems are placed. Such a facility must have specific installations, controls, and processes to be called a data center – otherwise it is simply a room with computers in it. Dedicated IT staff run those IT systems with specific processes to ensure proper functioning. The department that is responsible for data centers is often named IT Operations.

Even though data centers are not essential for high availability and disaster recovery, they provide good support for successful implementations. High availability and disaster recovery without data centers is possible, but requires much more effort. We will use this appendix to spell out in more detail what kinds of data center facilities will help to implement and retain high availability. Disaster recovery is supported by establishing or using a second data center at the disaster-recovery site, as described in Chap. 10.

From experience, there are four areas where errors are made most often and that cause incidents and outages:

1. **Changes** introduce errors and case failures.
2. **Air conditioning** does not work or is not sufficient.
3. **Power supply** is not successful.
4. **Fire protection** does not work (false alarm or erroneous release).

However, this is in well-run data centers. It is hard to estimate what error conditions would be the major root causes in data centers that do not follow the approach that is outlined in this appendix. For example, the potential error cause “tripping over a cable” does not appear in the list above because we do not have cables lying around that one could trip over. Therefore the list is a list of problems that one has to face in any case, even if one’s preparations are good.

Data center facilities are the best form of the *physical environment* that we have met so often in the previous chapters. These are rooms and equipment that are dedicated to the task of running IT systems properly

for a long time, with few outages. This appendix will not be a complete guide for how to build and operate a top-notch data center; that is beyond the scope of this book.

Broadly, data center facilities can be put up in the following categories:

- **Room installations** refer to issues around raised floors, cabling, and physical security. These issues are spelled out in Sect. B.1.
- **Heat and fire control** are crucial to keep computers operational and are described in Sect. B.2.
- **Power control** is another frequent failure area, and is handled in Sect. B.3.
- **Computer setups** are the way that computers are placed in data centers physically. This is the topic of Sect. B.4.

Let us have a look at each of those categories. But this section will provide only an overview of the most important topics, and only those that are especially relevant for high availability.

For example, the problem of grounding is not covered, even though it is a very important issue in many areas of the world. As well as the issue of water – there should be no water conduit in the room anywhere. There are more issues, and data center facility planning is such a big topic that it deserves a book of its own; we are just scratching the surface here.

B.1 Room Installation

Rooms in data centers are specifically equipped so that computer systems can be installed and run in them.

▸ *Raised Floors*

They have a raised floor system, where a second floor is installed above the actual floor. That second floor consists of removable tiles so that it is possible to access the gap everywhere. This gap is used for air circulation (as part of air conditioning, see later) and for cabling. Ideally, the gap between the base and the second floor is 60 cm (about 2 ft.), but it might be less depending on available room height.

The usage of tiles leads to the need to pay attention to the weight of computer systems and other equipment. Today the trend is towards ever higher levels of system integration. This often leads to more weight for the same volume and implies that we need to consider the impact that the weight makes on the ground at the system rack's feet.

Just as we have to pay attention to the impact of the racks, we also need to assure that the tiles that are on transport route are able to take the pressure. When one needs to reinforce the tiles at a specific place where a heavy computer system will be placed, this computer system

must be transported to this place somehow. If it is transported in one piece, the tiles on the way to this place might need reinforcement as well.

When your site has a raised floor, you probably also want to consider proper drainage. Otherwise, water damage – be it from pipes or rain – might damage the power and network connections that are usually placed into the underfloor space. For example, moisture alarms could be utilized to detect problems early.

▸ *Cabling*

Proper cabling is essential for high availability. There are quite a few failure causes that can be avoided by strict cabling policies:

- When cables are lying openly on the floor, somebody can trip over them and rip them off their jack, causing power outages, network outages, or others.
- When cable/jack connection are strong, tripping over them can even result in tearing out or overthrowing the connected systems.
- Long cables can be bent or damaged otherwise. Therefore it is better to use short cables to patch panels and have structured cabling between patch panels that can be easily tested and is redundant.
- Unmarked cables are often a source of erroneous installations. Even if the initial installation goes right, using unmarked cables or forgetting to mark them when their use changes is a sure recipe for an outage. Some day somebody will unplug the cables for repairs or upgrades, and will plug them in the wrong way.

The first thing that we insist on in a data center is *structured cabling*. This term means that cabling is integrated into the structure of a building and is independent of a specific server or specific system.[1] This is realized by four principles:

- Cabling between buildings is separated; there is a definitive transfer point between interbuilding cabling and vertical cabling.
- Cabling between building levels – *vertical cabling* – is separated; there are defined transfer points to interbuilding cabling and horizontal cabling.
  Vertical cabling is best done with fiber optics cables, because that creates a galvanic separation and prevents potential differences. (The main difference of potentials is in vertical distances.) With copper cable, there might be current on the grounding.
- Cabling on a building level – *horizontal cabling* – has clear transfer points to vertical cabling and system cabling.

[1] Though obviously cabling is not independent of the network technology used. Cabling for Ethernet, ATM, or Token Ring will be different, of course.

- System cabling is only over very short distances. It should be possible to check a system cable completely, i.e., to make optical checks without the need to open lots of trunks or floor tiles.
  Sometimes the transfer point from system cables to horizontal cabling of the data center is not directly beside a system, then the system cables come near other systems. That should be reduced as far as possible.

Cables should be tied: for horizontal cabling at least every 50 cm; for vertical cabling more frequently.

It is a big advantage to place information about cabling in one's configuration management database (Sect. C.3). Then one can always query the tool for the connectivity of systems and get initial information about dependencies and potential error causes.

Two independent network connections for a building also means two transfer points between interbuilding and vertical cabling, as well as redundant vertical cabling. There should also be two transfer points between vertical and horizontal cabling. Each row of system racks should have connections to both transfer points to create also redundant horizontal cabling, otherwise destruction of one transfer point would cause a connectivity outage of the whole floor. Within each system rack row, several patch panels provide transfer points between the redundant horizontal cabling and system cabling in short distances.

The transfer point between horizontal cabling and computer systems or active network components like big switches might have dozens, sometimes even hundreds of cables. It is mandatory to use an underfloor or overhead trunking system for such cabling to reduce the risks named before.

Such trunks must not be shared with power cables; there must be at least 6 cm between power and copper network cables. Use bridges to cross power and network cables, and always do so at a right angle.

▸ *Physical Security*

Major outages can be also caused by malevolent persons who get access to a computer's console – still access to the physical console has often more rights associated with it. There is also the risk of evildoers who damage computer systems physically with full intent in an act of sabotage.

Security and access control for a data center is usually established anyhow, for data security reasons. These measures are also valid precautions in the realm of high availability and will help to reduce the risk of physical sabotage. Video camera surveillance will enable faster detection of physical sabotage, or identification of the culprit after the fact.

For forensic analysis, one needs to know when somebody was in the data center and for how long, and on which systems this person worked in that time. Physical security must provide answers to these questions.

Server operations monitoring, often round-the-clock, is also sometimes integrated into physical security. For large data centers, these two functions are separate though. Even though the fully automated data center is a sought-after vision of the future, having human operators handle incidents and outages is still a valid precaution to reduce the risk to highly available installations. Appendix C presents more issues around proper operations processes.

B.2 Heat and Fire Control

Lots of computers and other devices in data centers produce lots of heat. This is so important that an acronym has been built around the need to handle this issue: HVAC for *heating, ventilation, and air conditioning*. With all this heat and with dangers of smoldering burns of electrical parts, fire is always an imminent risk in data centers.

▸ *Heat Control*

Computers, in particular high-end servers, are notoriously influenced by the temperature they operate at. Their electronic components produce so much heat that they will shut down in short time if nothing is done to reduce that heat. The heat must be dispensed and transported elsewhere.

Actually, typical IT systems will work for quite some time at higher room temperature. But even if they continue to work in warm areas, they age faster and the probability of failures will become higher (see the discussion of the bath tube curve in Appendix A.6).

Some time ago, computers had water cooling systems. They did not heat up the room themselves. Then computer systems got air cooling. Early systems relied on data centers where cold air is blown out of the floor and they emitted hot air at the top, where an air conditioning system could handle it. Nowadays, vendors build computers as they like: they ingest cold air where they want and blow out hot air where they want (most of the time not at the top).

This leads to the requirements:

- Where cold air is sucked in, there must be a cold air current.
- Where hot air is blown out, there must not be a cold air inlet of another system.

It might well be that future systems will deploy water cooling again. The trend towards ever higher system densities per volume creates heat problems as well. And water cooling needs less energy than air conditioning. Since the energy for air conditioning costs easily more than the whole server, this is an interesting aspect that would save ongoing costs and raise system availability.

Ventilation is used to dispense heat evenly within a room, but will *not* cool it. Since humans feel the windchill factor more than the real temperature, it is a high risk to depend on one's senses for room temperature. Instead, good-precision air conditioning should be used to cool down the whole room to roughly 18°C (about 64°F).

Good precision here means that the air flow in the room is observed and planned. There are sprays that emit a kind of fog that can be used to visualize air currents and help to analyze why a computer system is getting too hot. Of course, first of all one has to measure both the temperature in the computers and the temperature at multiple places in the room. Without that information one would not even know that a system gets too hot or that there are hot spots in a room.

When new computers are installed in a data center, it must ensured that the air conditioning system has enough capacity for the new equipment as well. Otherwise, availability of all systems in the data center is endangered, not just that of the new equipment.

This is an important consideration as new vendors market new systems with ever-increasing packing density, e.g., blade servers. Often, racks are planned and assembled by the IT staff and not by vendors. If one has not very strong air conditioning, one needs to pay attention to the heat production and dispensation for such dense systems. Sometimes it is necessary that one does not fill up the rack completely, but leaves some space for air circulation and less heat production.

Air conditioning is a technical system like any other, and is bound to fail as well, i.e., air conditioning can well be the single point of failure in a data center. To combat this, a backup air conditioning system should be in place, or the different components of the air conditioning should be redundant. As with all redundant technology, one needs to manage the redundancy, e.g., to monitor that the components run at all and exchange or repair them otherwise.

Such monitoring is done in a HVAC control environment. Best-of-breed installations make that control environment redundant again, with manual management in the case of failures. We might also attach the HVAC environment (both the control component and the actual air conditioning system) to the uninterruptible power supply (Sect. B.3), to protect the heat control against power outages.

But HVAC monitoring does not consist of room-temperature sensors alone. Most computer systems allow the temperature of the CPU or fans to be queried. Computer monitoring should observe such temperatures as well, with properly defined escalations when the temperature is becoming too high. (Data sheets of hardware components usually tell about their typical and best operating temperature.)

Good working air conditioning is sine qua non for high-availability installations in data centers: without it, the servers will not run.

▸ *Fire Protection*

Since computer equipment produces heat in exuberance, and since any electrical component is in danger of smoldering, fire is one of the most probable risks to the physical environment. Therefore, good risk mitigation strategies call for precautions to detect fire early and also to fight it without damaging all the equipment.

A very important facility is early-warning fire and smoke detection systems. They can often detect problems before fire actually breaks out. Better systems even come with sensors to do spot detection, i.e., to detect the place in the room where the smoke comes from. Sometimes they measure particles given off by hot components and trigger alarms if those particles pass thresholds. With such fire prevention systems, one can often turn off problematic devices before an actual fire breaks out.

Still, that device will be unusable from now on. But that is the known area of high-availability setups: there will be redundant components that can take over functionality. This is one example where internal monitoring of any computer component is not sufficient. Sure, we would notice that the component gets too hot and might be able to disable it. But if it has started to smolder already, disabling the component might not be sufficient; instead one has to turn it off physically, maybe even remove it from the room as soon as possible.

The early-warning fire protection system is only the first line of defense and it would be foolish to rely on it alone. The fire could break out faster than our reaction time, or it could be a spark-triggered fire without slow buildup of smoke. Then we need fire extinguishers or a fire suppression system.

Conventionally, fire suppression systems are water sprinkler systems. These are not of much use in data centers, as they damage the electrical equipment beyond repair. The same is true for foam-based fire extinguishers – if anybody uses water or foam in a data center, the chances are high that disaster recovery for the whole site will be tested in real time soon after.

Instead, data centers utilize fire suppression systems that push out or reduce oxygen in the air, so that fire cannot spread anymore. Originally, CO_2 was used, flooded from the floor. This flooding creates mechanical pressure that is so high that it damages cabling. In addition, it is acidic; with water it forms carbonic acid. Of course, it is also toxic, but almost all fire suppression materials are toxic, so that is not a differentiator.

Then halon was used often, a gas that pushes oxygen out of the room. But if people remain in the room without emergency breathing oxygen supplies, or if they are unconscious, usage of a halon-based system is often deadly. (Well, smoke and fire themselves are also often deadly.) But halon is also one of the chemicals that destroys stratospheric ozone; therefore

it was banned in the EU, and in the US the EPA strongly encourages the use of non-ozone-depleting alternatives.

Today, halon alternatives like *FM-200* or *Inergen* are available. FM-200 belongs to the class of halocarbon agents that extinguish fire primarily via the absorption of heat, whereas Inergen is an inert gas agent (it consists of a mixture of argon and nitrogen) that works via oxygen depletion.

Last, but not least, fire protection zones are valuable. A fire extinguisher can operate only in those areas where fire was detected. Physical fire containment strategies are part of such a strategy and should be considered too. Fire doors or other physical firebreaks can be used to contain fire within a small physical area and confine damage to some of your highly available systems, and protect against spreading to all of them.

B.3 Power Control

Other frequent sources of computer failures are power outages or fluctuations in power frequency and power strength. It is mandatory for any data center to protect against these failures.

As a basic level of protection, we establish *uninterruptible power supplies* (UPS). These are devices that are responsible for maintaining a continuous and stable power supply for the hardware. UPS devices protect against some or all of the nine standard power problems:

1. Power failure
2. Power sag (undervoltage for up to a few seconds)
3. Power surge (overvoltage for up to a few seconds)
4. Brownout (long-term undervoltage for minutes or days)
5. Long-term overvoltage for minutes or days
6. Line noise superimposed on the power waveform
7. Frequency variation of the power waveform
8. Switching transient (undervoltage or overvoltage for up to a few nanoseconds)
9. Harmonic multiples of power frequency superimposed on the power waveform

Good data centers never connect any computer equipment directly to the power grid, but always place a UPS in-between. This it at least true for all highly available systems. A *power distribution unit* (PDU) connects the UPS devices to the computers.

As for every other device, UPS systems may be defective themselves, as well as PDUs; therefore it is necessary to monitor them all the time to detect failures early. Detecting UPS failures during power failures is bound to lead to major outages.

Installation of new hardware in the data center always has to be accompanied with an analysis of power demand changes. It must be checked that the UPS will handle changed power requirements, otherwise it must be upgraded as well (which can be quite expensive, by the way). Otherwise, the new hardware will pose a risk to all other installations in the data center; e.g., installation of an unrelated business foundation server could damage our highly available mission-critical server and lead to a major outage.

UPS devices contain batteries and will protect against short power failures in the range of a few minutes. If a power outage takes longer, one must either start disaster recovery and switch to a disaster-recovery site, or one needs to have backup generators (e.g., diesel generators) that generate power themselves.

Shortly before the batteries are dead, UPS management software will trigger automatic shutdown of attached systems since proper shutdown is much better than crashing the systems – at least the data will be in a consistent state. For high availability this might have consequences beyond the actual service: if some other system depends on this service, its availability will be affected as well; and if the dependent system is a failover cluster, it may well lead to perpetual failovers in the cluster.

Another, but more expensive, possibility to handle power outages is to get two independent power grid source connections. Then all elements of the electrical system are redundant, all servers are connected to both grids; outage of one power grid will be handled by the second grid. But we must be realistic and must see that this is very hard to realize. The problem here is not to get two power connections – any sales person of any utility company will sell that to you. The problem is the innocent term *independent*: real independence is seldom in power grid structures, and in accordance with Murphy's Law the excavator will rip apart the big grid cable in the street that is used for both input sources.

By no means does that mean that we should not strive for redundancy in our power supplies and also our power grid source connections. We can test for functional redundancy of our own installations, but will not be able to test the redundancy of grid connections.

Redundancy for internal power supplies, for UPS systems, and also for all connections is necessary for several situations. First of all, these are hardware components that wear with usage and where the probability of failures is quite high. Therefore it is advisable to protect against such failures at the cause and add redundancy already there. In particular, UPS devices use batteries that have only a limited lifetime.

Second, redundant power installations allow repair activities without disturbing the actual system usage. Maintenance work at the UPS can be done as needed if one has another UPS system that is used at this time. The same holds for power connections and power supplies.

Yet another issue to consider is the effect of powering off and on of computer systems. We need a dependency list of computer systems just

for powering them off cleanly: the dependency graph shows us which systems are independent and can be shut down first. Otherwise, a cluster component could notice the shutdown and could trigger failovers of dependent components. The same holds for power-on sequences.

Powering on must be done carefully for other reasons. During power-on, most components draw much more power than in normal run situations. This can bring our overall power supply and the UPS to their limits. Therefore one should not turn on all devices at the same time. Instead, one should start them in a preplanned sequence, to play it safe both with power consumption and with service dependencies.

In summary, power outages are a risk with a high probability for highly available systems. Protection against power outages is possible, though expensive if one goes the last mile and utilizes a backup generator. At least, usage of UPS devices is mandatory, as is monitoring those devices.

B.4 Computer Setup

At first glance, the way that computer systems are physically set up is not relevant for high availability. It does not matter for the functionality if a server is a white-box medium-tower system or if it is mounted in a rack cabinet. But that first glance only looks at the failure situation and ignores the repair situation.

We should place our servers into proper rack cabinets (mostly industry-standard 19" racks). This leads to better cabling, no traps by cables or boxes, stable ventilation, and good access to the system plugs. Proper labeling of each system helps to locate it quickly in the case of necessary repairs.

Be careful when you mount systems into a rack. Racks can totter or even tumble if you pull out systems at the very top. This can have disastrous consequences for other systems in that rack and destroy their availability, i.e., maintenance work at a system with low priority can influence availability of a mission-critical system if they are in the same rack.

Such rack cabinets, as well as mainframe-type computer systems and big storage units are placed into rows in a data center. Such proper room layout also helps to identify computer systems quickly in the case of emergencies.

All these precautions will reduce the mean time to repair. One is able to locate a failed system quickly, one can access the system's components properly, and one has ordered and labeled connections that will be less probably reconnected erroneously.

Remember the start of this book – mean time to repair is the setscrew that is best to turn to raise our overall availability, as the mean time between failures cannot be changed as easily.

C

Service Support Processes

This book focuses on the principles, architecture, and implementation of high availability and disaster recovery. As such, it covers 90% of the projects that introduce those capabilities in IT environments. But as everybody knows, there is another 90% to get the job done: run the systems without deteriorating them. This is the job of the data center staff and of the system administrators who manage and maintain the systems.

The importance of the run-the-systems phase to retain high-availability and disaster-recovery capabilities is generally accepted but often not emphasized enough.

> **No tool and no automation can save you from bad operations – bad operations will make any good design fail. It is the biggest threat to high availability.**

There is a de facto standard for managing IT operations, called the *Information Technology Infrastructure Library* or ITIL for short. ITIL is a process-centric view that gives guidelines for best-practice data center management. ITIL was created[1] and is owned by the Office of Government Commerce (OGC) of the UK's Treasury.

ITIL defines the task of data center management in five areas:

1. The business perspective
2. Managing applications
3. Deliver IT Services
4. Support IT Services
5. Manage the infrastructure

IT service delivery [7] is at the heart of this whole book since it contains the descriptions of *availability management*, *service level man-*

[1] Actually, ITIL was created in the late 1980s by the Central Computer and Telecommunications Agency (CCTA), which was merged into OGC in April 2001.

agement, and *IT service continuity management* processes. We have met these processes several times throughout this book.

IT service support [6] is an area that has relevant material for the scope of this section. The area contains process descriptions of:

- Service desk
- Incident management
- Problem management
- Configuration management
- Change management
- Release management

When we apply these processes, we will have a very good chance that our run-the-systems job is improved and that the systems will stay highly available and disaster recovery will remain possible.

In the rest of this section, we will have a look at all those processes except *service desk*.

C.1 Incident Management

In ITIL terminology,

> **An 'Incident' is any event which is not part of the standard operation of a service and which causes, or may cause, an interruption to, or a reduction in, the quality of that service.**

The goal of incident management is to restore system functionality (as defined in the service level agreement) as quickly as possible.

Such an event may be a an application or hardware failure, or it may be a service request. Incident management is engaged in detection and recording of incidents, classification, initial support, and resolution.

The focus of incident management is the system's user. Its objective is to get the user working again as quickly as possible. Workarounds are perfectly valid resolutions; the underlying root cause of the incident may not be known and there is often not enough time to research it.

The cause of incidents may be obvious and can be remedied in short order. Then no further action is needed. But when the underlying cause is not known, one should try to find a workaround and transform the incident into a problem, see Sect. C.2. It is difficult to decide when this propagation is done. If it is done too often, resources will be bound that could improve the IT service otherwise (or could make operations cheaper). If it is not done often enough, structural problems are uncovered and symptoms are fought all the time in incident management, damaging service availability.

This is an important connection between incident management and high availability that is often ignored. One can learn a lot by looking at incidents that are communicated by the system's users or are triggered by automatic monitoring systems. Often these incidents give new material for failure scenarios of future projects, or they cover up deeper-level problems that should be addressed to improve availability.

But why do incidents happen at all? At first sight, high availability should ensure that no incidents happen; all are handled by the redundant systems. But that is not the case in practical terms:

- When a component fails, it must be repaired. While this might be considered as "part of the standard operation," many organizations choose to handle such failures also via their established incident management processes. After all, during the duration of repair, the missing redundancy (or the repair itself) may cause interruption of the service.
- We have seen several times that redundancy of components must be managed somehow. Time and time again we used the formulation that it may be managed by administrators. But how is this done in practical terms? Operational procedures are the answer, and here incident management enters the room. It is an established process and procedure infrastructure that can trigger resolution of failure by manually establishing the redundant component.
- Murphy's Law is true. Anything that can go wrong, will go wrong *over time*.[2] Even if the project's implementation has the best analysis, reality will bite and there will be incidents in areas where experience was not sufficient, or assumptions were wrong, etc.

In the realm of high availability and disaster recovery, it is also relevant that *escalation procedures* are part of the incident management process. Such escalations are needed when the incident cannot be handled in the normal service level agreement repair times and when it evolves into a major outage.

C.2 Problem Management

Problem management is concerned with root cause analysis of failures, and with methods to prevent them in the future. The ITIL documents define the topic of the study as follows:

> **A 'Problem' is a condition that is either identified as a result of multiple Incidents that exhibit common symptoms. Problems can also be identified from a single significant**

[2] Murphy's Law does not mean that every possible failure happens immediately. It means that every possible failure happens *eventually*.

Incident, indicative of a single error, for which the cause is unknown, but for which the impact is significant.

Problem management is intended to reduce the number and severity of incidents, and therefore proper implementation of this process is of high importance for high availability.

Root cause analysis (RCA) of outages and failures is the first step for continuous improvement of an IT system. Its result should:

- Enable incident management to resolve incidents faster by providing a database of known errors and appropriate resolutions
- Improve IT system implementation and operations with changes that prevent such problems in the future

This means that the goals of problem management are twofold: with the support for incident management we try to reduce the needed mean time to repair, which increases our availability. With changes to the installed systems that prevent such problems, we increase the mean time before failure, also with the result of increasing our availability.

Problem management is not only triggered by incidents. Technicians often detect problems during their normal day-to-day work. They stumble regularly over installations or configurations that *do not look right*. Such preliminary estimations are quite often correct and are grounded in solid experience. We must not forfeit that experience and must incorporate it into proactive problem management, i.e., instead of adding a "fix me" comment in some script that will soon be forgotten, system administrators should be encouraged to log the issue in a problem management database for later analysis by peers. It is best to frame such encouragement in real processes: establish metrics as to who discovered and analyzed successfully the most problems, and give out awards to your employees who are not too shy to name areas where improvements are needed. Such proactive problem management is a very valuable tool for improving availability and asserting the success of disaster recovery.

One of the biggest problems with problem management is the tendency that root cause analysis results are not followed through. Time and time again it happens that a problem is discovered and analyzed, the root cause is found, but no change is triggered owing to the inertia of the IT system that is running. "Never change a running system" is a very good approach at first, but there are limits to it. When known errors exist and cost/effort-benefit analysis for a change is positive as well, the high-availability objective should take a front seat and should enforce the change.

C.3 Configuration Management

Configuration management is concerned with management of the knowledge of our systems. Its central term is the configuration item (CI) that ITIL defines by example. Their example can be abstracted into the following definition:

> **A CI is a hardware, software, service, or documentation asset, or one of its subcomponents that is independently managed. A CI has a type, a unique identifier, and arbitrary relations to other CIs, the most important are "consists of" and "uses." A CI may have attributes (key-value pairs) that describe it further.**

You will immediately have noticed that this definition is very similar to our *component* term that we introduced in Chap. 3. So here we find a standardized process to manage components that we need to analyze systems.

Configuration management collects information about system components. It tracks actively changes and builds a knowledge base that can be used by other ITIL processes and must be kept up to date by other processes too. This knowledge base is named the *configuration management database* (CMDB).

The CMDB should keep all information:

- About configuration items
- Their attributes
- Their relationships
- Their version history
- Their configuration
- Their documentation

Sometimes, configuration and documentation are simply seen as attributes; sometimes they are mentioned to emphasize their importance.

This sounds to good to be true, doesn't it? Here we have the perfect source and storage for our architecture components, the dependency diagrams, and the redundancy information that we need so urgently in high-availability and disaster-recovery projects. Figure C.1 on the next page illustrates that goal.

Well, the reality is that existing CMDBs seldom fulfill the need of such projects, and that has several reasons, as will be shown now. A very important part of configuration management is the decision about the granularity of CIs. That decision is influenced by two factors:

1. In the ITIL model, only CIs can have relations, namely, to other CIs. It is not possible to have a relation between or to attributes. That is what distinguishes them from attributes, otherwise every attribute could be a CI itself. (The attribute name would be the CI's type, the

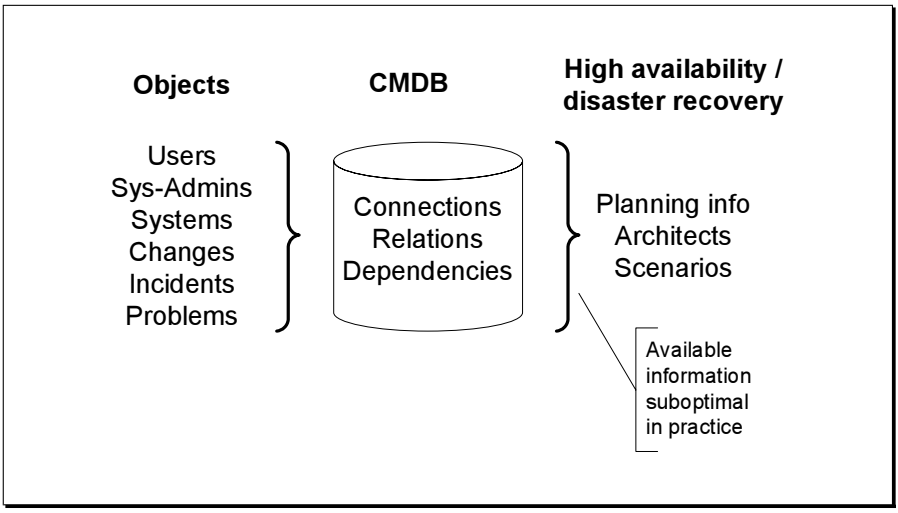

Fig. C.1. Theoretical usage of the configuration management database (*CMDB*) for high availability and disaster recovery

attribute's value would be the CI's identifier, and an "is-described" relationship would exist between the CI and these new attribute CIs.) That is, when we want to express relations in the CMDB, we need to make that component a CI. When we do not want to express relations, this component can often be modeled as an attribute.

2. CIs are the preferred unit of management by other ITIL processes, in particular change, incident, and problem management. Of course, they can and will refer to CI attributes as well, but that is more clumsy than referring directly to CIs. Depending on their demands of granularity, the CMDB model will be coarse or very fine grained.

As an example, a network interface card (NIC) may well be an attribute of a CI server and its own CI. Both are valid models:

- If change and incident management only refer to servers and describe the error message according to the server, noting the NIC existence (maybe with associated configuration information like the MAC address) is sufficient.
- Treating the NIC as its own CI allows us to express its relationship to other parts, e.g., a redundancy relationship to another NIC. It also allows us to express the exact components that are addressed by changes. On the other hand, that makes change descriptions much harder to build as one often does not know up-front what components a change will address. In fact, keeping fine-grained CIs consistent, up to date, and with a sensible history that can be assessed by people is *very* hard.

Table C.1. ITIL objectives for CMDB (excerpt). *CI* configuration item

ITIL process	CMDB support
Incident management	Automated identification of other affected CIs when any CI is the subject of an incident.
	Identification of owners, responsible IT staff, vendor contact information, support contract details
Problem management	Integration of problem management data within the CMDB, or at least an interface.
	Easy interrogation and trend analysis reports
Change management	Identification or related CIs affected by proposed change to assist with impact assessment.
	Record CIs that are affected by authorized changes.
	Maintenance of a history of all CIs.
	The ability to show graphically the configuration and input information in that graphical representation.
	Enable the organization to reduce the use of unauthorized software
Release management	Record baselines of CIs and CI packages, to which to revert with known consequences.
	Register CI status changes when releases are implemented.
	Support roll-out across distributed locations by providing information on the versions of CIs and the changes that are incorporated into a release

But our dependency diagrams that we use for high-availability and disaster-recovery design need to express their redundancy and thus needs them as CIs. Usual CMDB data schema modeling happens independently of high-availability or disaster-recovery projects, is oriented towards the need of other ITIL processes, and does not take our demands from that area into account. Therefore, the chances are low that the CMDB is really a good place to store our complete dependency and redundancy data.

According to the ITIL, the CMDB is the all-singing all-dancing data store that answers every question about our systems and their configuration. Table C.1 lists just a few of the intended usages that are already overwhelming.

Add the point that industry pundits agree that a single central CMDB is not feasible and federated CMDBs will be the way of the future. Though they do not explain in technical details how such a federation of CMDBs is supposed to work and how the CMDBs are supposed to be interlinked. (Vaguely mumbling about “Web Services” should not be seen as an explanation.)

In practice, many existing systems are quite crude and not as interlinked as the theory wants it. This may change in the future as more and

more data centers orient their management procedures towards the ITIL. But good CMDBs are hard to establish, cost a lot of effort to maintain, and their benefits are not immediately visible. They are also a technical measure, and most ITIL introductions concentrate on the management and process side and leave aside radical technical improvements.

This reflection leads to the point that while configuration management and especially CMDBs could contribute a lot to our work on high availability and disaster recovery, it is not probable they will do so in the short term to the midterm.

C.4 Change Management

Good change management practice is essential to keep high availability and the ability to do disaster recovery. Change management is the process to control changes in your IT environment, be they caused by change requests or by problem management. As the ITIL says,

> **Change is the process of moving from one defined state to another.**

Changes to your systems should be:

- Proposed with justifications; change requests without specifying cause and effect in technical details are not acceptable
- Reviewed by subject-matter experts if they introduce new single points of failure or new dependencies, to make sure that high-availability or disaster-recovery capabilities are not damaged
- Checked to see if any change in a primary system is accompanied with a change on the respective disaster-recovery system
- Approved by a *change advisory board* that evaluates their business impact and coordinates the changes with other changes

With proper change management in place, the exposure to risk is minimized. For our highly available systems and our disaster-recovery precaution, any change is surely a risk and may be a chance. This cannot be emphasized enough: *every change is a risk to availability*. A new functionality or a new component is introduced into a running IT system. Both the migration to new functionality or new components, and the new components themselves may not work. Testing will show where the functionality works, but not where it may fail. As Murphy's Law says, "anything that can go wrong, will go wrong over time."

It is thus very important that every change comes with a back-out plan, a method to revert any changes to the starting point. This back-out plan is the last resort when something goes wrong and the risk of a change manifests itself as a real failure.

On the other hand, the risk of change can be worth it. If it increases reliability of the system overall, we have exchanged the increased short-term risk of change against the decreased long-term risk of failures. That is what the reviews are for, to analyze a change request and judge the associated risk and benefit. Experience is needed for such analysis and judgment; only many years of seeing both good and bad practice gives this ability.

For mission-critical highly available servers, it is mandatory that *every* change is reviewed in detail by a senior subject-matter expert. For servers in the business-important or business-foundation categories, it is often sufficient to do summary reviews of changes and dive into detailed reviews for changes with more impact.

When you do a change to a server with disaster recovery in place, do not forget that you might have to update your Disaster Recovery Emergency Pack as well. Printing out up-to-date information is often forgotten, as it is inconvenient and seen as inefficient. But when a major outage occurs, one is glad to have material that is independent of systems that might be destroyed.

C.5 Release Management

Release management is the process to bring new software, hardware, or configurations into production. According to the ITIL,

> **A *Release* describes a collection of authorized changes to an IT service. It is a collection of new and/or changed configuration items which are tested and introduced into the live environment together.**

For highly available systems, this is an extremely important area. Releases must be identified, planned, and must be tested thoroughly before they are put into production. Together, change management and release management are the crucial processes that ensure ongoing availability and serviceability of IT services:

- Change management makes sure that the organizational aspects of changes – coordination with business owners, announcements, reviews, and approval – are done right.
- Release management ensures that the technological aspects of changes – planning, tests, and roll-out – are done right.

Release management has several important duties:

1. To determine the release unit
2. To testing the release unit
3. To roll out the release unit

First, we need to determine which changes are released together. This set of changes is called the *release unit*. As so often with such processes, that decision is a matter of experience. The change set must neither be too small nor too large. If it is too small, we have lots of releases and every release has associated risks that are independent of its size (e.g., the roll-out might not work). We do not want to introduce a new release for every minor change. On the other hand, if a release becomes too large, we have the issue of many independent changes that can go wrong and have often unknown interdependencies. Proper testing of large releases is very difficult and therefore error-prone. A release unit should be large enough to qualify for an improvement of the IT service, but should focus on a few improvements and should not lump together many changes from different areas.

When a release unit is tested, it must be ensured that not only functionality tests are done. This is the most common error in release unit testing: only the working functionality is tested, not the failure situations. That is, there is a description of what a change is supposed to do. Tests most often determine if the changed IT service really has the new functionality, and if the old unchanged functionality still works. In addition, back-out plans must be tested, we will look at them again later.

But it is as important to test how the new components react in the case of failures. For example, if we have utilized a new application in a failover cluster, does failover still work? In such a situation, it is not only important to start the new service on one cluster node and see if it works. It is as important to cause several failovers, abort the processes, maybe damage the data store, and check if restarts, service switches, and data consistency checks still work. Similar tests must be done when new hardware is installed. For example, a new storage unit must be thoroughly checked that redundant cabling is really done correctly, etc.

When hardware changes are rolled out, it must be checked again (this check is also done during planning) that the new hardware does not disturb the disaster-recovery concept. For example, adding new storage to the primary site usually means that we also need to add it to the disaster-recovery system. Also, replication must be established for data on the new storage system, or it must be checked that the available replication still works.

It is best to have tool support for roll-out of software releases. Installing software without package management or other software distribution tool support is a recipe for disaster. Only with tool support is it possible to check which software is installed on a system in which version, or which software a file belongs to. Both are essential capabilities that are needed during incident and problem management. Sadly, current software release tools lack support for cluster environments. Here experienced IT staff have to augment vendor-supplied mechanisms with

their own developments and own procedures to keep control over their systems.

Please note that the requirements for software distribution support depend on the target system. There is a vast difference if one wants to roll-out a new Office version to a few thousand desktops, or if one installs a new Oracle release on a mission-critical server. While the first can and should be automated and the distribution should utilize a push model from a central distribution server, utilizing a fully automated push model for an Oracle update is a definitive way to get a major outage. Here the installation must be done on the server, and tool-supported installation steps are intermixed with manual checks if functionality and performance are not impacted.

As in change management, a back-out plan is an indispensable part of any release unit. The back-out plan must spell out both the organizational and the technical details of how one can roll back to a previous state. If that is not possible for technical reasons, the back-out plan must describe how a spare component can take over the service (e.g., the disaster-recovery system can be utilized). It is also necessary to specify the *point of no return*. This is the point in time when a roll back cannot be done anymore because it would need too long. At this time, the actual state of the release roll-out must be reviewed and a decision must be made as to whether one will continue successfully or whether it is better to roll back and start anew at another date.

C.6 Information Gathering and Reporting

A German proverb says, "Do good things and tell about it." This is also true for any activity in data centers. A big problem for business owners, executive managers, and fellow colleagues is that they do not know what work has been done in the past and what is planned for the future.

An important sign of a well-run data center is that information is gathered that provides metrics about the work that is done and the state of the IT system. That information should be both technical and business-oriented. As an example, it is good practice to collect information about:

- Changes
- How many incidents there are for a system
- How many changes fail
- How many changes lead to problems or incidents in a short time span
- Which releases are out, and the history of releases
- The rate of change for incidents, problems, and changes
- Failovers
- How many outages there are, and for how long
- How many and which IT systems are affected

Of interest is also a more complex analysis that gives hints to improve change and release processes. For example, consider the statistics "how many changes fail or cause problems, depending on the number of changes in a given time range." When we have ten changes at a weekend, and one of them fails – how many will we have when we have 50 changes at a weekend? The answer cannot be generalized and depends greatly on the available staff and available skills. But most often the relation is not linear, and more changes may lead to more or less error, relatively speaking.

One can improve one's statistics by creating adequate categorizations. Both change and problem categories are important. For example, let us assume that we have the (simple) change categorization:

- New installation
- Function change
- Incident repair (emergency change)
- Problem repair
- Security upgrade

Even with such a simple categorization, we can already imagine that the relation of errors to change categories is very interesting and can lead to data that tells us where we need to improve our processes, to achieve lower failure rates and thus higher availability. When the problems and incidents are also categorized (e.g., according to how many people were affected), then we can give even better input to process improvement proposals.

It is best practice to generate several reports for different target audiences that differ in information selection and level of details. For example, a report for a business owner will present different information from a report for the operations manager or the operations staff. The technical reports should always contain the information that is sent to the business owner. If she or he phones in and demands information for a specific item in her or his report, this information should be available in the operations manager's report as well.

For executive management, establishment of a *dashboard system* is sensible. Such a system collects information from many technical and business systems and condenses them into key performance indicators for essential business and IT services. Such information is typically presented on a very high abstraction level (e.g., "green/yellow/red traffic lights" for essential services) to compress the overall state of IT into one short table.

Better executive dashboards are tool-supported and also give the ability to "drill down" to more detailed information, maybe with a link to the technical information systems for the executives who are interested in technical details.

References

[1] Brilliant SS, Knight JC, Leveson NG (1990) Analysis of faults in an n-version software experiment. IEEE Trans Software Eng 16(2):238–247, DOI http://dx.doi.org/10.1109/32.44387

[2] Brooks FP (1975) The Mythical Man-Month. Addison-Wesley, Reading, NJ, USA

[3] Gibson GA (1992) Redundant Disk Arrays: Reliable, Parallel Secondary Storage. ACM Distinguished Dissertation, MIT Press, Cambridge, USA, also published twice at University of California, Berkeley, in 1992 and again as Technical Report CSD-91-613 in 1999

[4] Littlewood B, Popov P, Strigini L (2001) Modeling software design diversity: a review. ACM Comput Surv 33(2):177–208, DOI http://doi.acm.org/10.1145/384192.384195

[5] Murphy J, Morgan TW (2006) Availability, reliability, and survivability: An introduction and some contractual implications. STSC CrossTalk 3:26–29

[6] Office of Government Commerce (2000) ITIL Service Support. The Stationery Office Books, London, UK

[7] Office of Government Commerce (2001) ITIL Service Delivery Manual. The Stationery Office Books, London, UK

[8] Sowa JF, Zachman JA (1992) Extending and formalizing the framework for information systems architecture. IBM Syst J 31(3):590–616, IBM Publication G321-5488

[9] Spiegel MR (1975) Probability and Statistics. Schaum's Outline Series, McGraw-Hill, New York, USA

[10] Tanenbaum AS, Steen MV (2001) Distributed Systems: Principles and Paradigms. Prentice Hall, Upper Saddle River, NJ, USA

[11] Tobias PA, Trindade D (1995) Applied Reliability. Chapman & Hall/CRC, New York, USA

[12] Toigo JW (1989) Disaster Recovery Planning: Managing Risk and Catastrophe in Information Systems. Prentice Hall, Upper Saddle River, NJ, USA

[13] Trindade D, Nathan S (2005) Simple plots for monitoring the field reliability of repairable systems. In: Annual Reliability and Maintainability Symposium (RAMS), IEEE, Alexandria, VI, USA, pp 539–544
[14] van Lint JH (1982) Introduction to Coding Theory. Graduate Texts in Mathematics, Springer, Berlin Heidelberg New York
[15] Zachman JA (1987) A framework for information systems architecture. IBM Syst J 26(3):276–292, IBM Publication G321-5298

Index