Empirical Estimates of Software Availability of Deployed Systems

Audris Mockus
Avaya Research
233 Mt Airy Rd
Basking Ridge, NJ 07920
audris@mockus.org

ABSTRACT

We consider empirical evaluation of the availability of the deployed software. Evaluation of real systems is more realistic, more accurate, and provides higher level of confidence than simulations, testing, or models. We process and model information gathered from a variety of operational and service support systems to obtain estimates of software reliability and availability. The three principal quantities are the total runtime, the number of outages, and the duration of outages. We consider methods to assess the quality of information in customer support systems, discuss advantages and disadvantages of various sources, consider methods to deal with missing data, and ways to construct bounds on measures that are not directly available. We propose a method to assess empirically software availability and reliability based on information from operational customer support and inventory systems and use a case study of a large communications system to investigate factors affecting software reliability. We find large variations among platforms and releases and find the failure rate to vary over time.

Categories and Subject Descriptors: D.2.8 [Software Engineering]: Metrics — Product metrics, Management – Software quality assurance (SQA)

General Terms: Measurement, Reliability

1. INTRODUCTION

Many businesses depend upon uninterrupted performance of software systems to support their activities and any unplanned downtime of such systems results in significant costs and negative publicity. Some common examples include trading and banking systems. Therefore, many software and hardware systems are specially designed to support such high availability applications. A lot of effort and thought has gone into design and verification activities to improve the reliability of such products. The modeling of fault occurrences is also a large and important field of study. Unfortunately, few publications are devoted to an empirical evaluation of actual availability of deployed systems or on studies how to collect such data. Evaluation of real systems is more realistic, more accurate, and provides higher level of confidence than alternative approaches that include simulation, testing, and quality models. Furthermore, some customers were requesting availability measures, therfore we set to obtain the relevant data and estimates in the context of high availability communication software for traditional and IP telephony. Such estimates of reliability and availability are also commonly referred to as deployed or operational measures of reliability and availability.

We used information gathered from a variety of operational and service support systems and then archived, validated, filtered, modeled, and integrated that data to obtain the quantities needed to estimate the reliability and availability of the deployed systems. The three principal components were the total runtime of the relevant systems, the number of outages, and the duration of outages. Perhaps not surprisingly, the outage information was recorded much more carefully than the information about the system population, and especially, the population that is capable of reporting the outages.

We consider methods to assess the quality of information in customer support systems, discuss advantages and disadvantages of various sources, consider methods to deal with missing data, and ways to construct bounds on measures that are not directly available.

Our primary contribution is to propose a method to assess empirically software availability and reliability based on information from operational customer support and inventory systems. In addition, the novelty of our approach has several aspects. The precise information about the system population, configuration, and age is linked to the outage information in order to produce more accurate estimates of availability. The methodology of data collection to estimate availability of software is proposed. The experiences and findings applying the approach to a large enterprise communication system are discussed. We ask several practical and theoretical questions and evaluate them based on the obtained results. In particular, we compare simple to obtain approximations of reliability with more accurate, but harder to obtain estimates. We also evaluate if the common reliability measure of mean time between failures (MTBF) is appropriate for varying system runtimes by investigating the hazard function.

The findings from a case study show that the software reliability strongly depends on several of factors, and the failure rate is not constant over time. This suggests that the his-

ISESE'06, September 21–22, 2006, Rio de Janeiro, Brazil.

toric measures that imply a constant failure rate (including MTBF) may not be appropriate.

Our general approach contains four main steps:

1. Events representing outages are collected and associated with the systems that generated them.
2. Outage durations are estimated.
3. The runtime for systems that can potentially generate such events are estimated.
4. The quality models are fit for our case study and the relevant results are reported.

We start with motivation in Section 2, describe the context and related work in Section 3, describe various sources of data in Section 5, present the methodology in Section 6, describe the results in Section 7, and discuss the approach in Section 8.

2. MOTIVATION

From our experience it appears that the lack of results in the area of empirical estimation of software availability is at least partly attributable to a large amount of time and effort needed to empirically estimate the availability of deployed systems [1]. Another issue may be the diversity of tools and processes used to track customers issues and deployed inventory. This makes it difficult to propose a methodology general enough to apply widely. The resulting lack of publications leaves a gap in methods, metrics, and benchmarks needed to collect, validate, and compare the empirical availability measures. Nevertheless, the potential benefits are great. Despite careful design and extensive testing and verification, issues do occur in deployed systems. The models, simulations, and even early introduction programs do not accurately represent actual experiences of the customers running the system partly due to a large diversity of configurations and usage patters end users tend to subject the software.

Furthermore, such high-availability systems tend to be deployed earlier by customers with lower demands on availability. The availability estimates based on such early deployment can then be used to improve systems for the most sensitive customers or to adjust the deployment schedule [11].

The ability to gauge the actual availability of deployed systems can provide assurances to customers and action plans for software producers. Such action plans may be focused not simply by the values of the availability measures, but also by the relative contribution of different outage causes that could be gleaned from the predictions of reliability and availability models.

3. BACKGROUND AND RELATED WORK

One of the most important aspects of assessing availability of deployed systems is to obtain precise information about dates and configurations of software installations and upgrades and being able to link that information to reported problems. There may be two major issues in being able to obtain such data. The lack of information on who has installed what software and when may be unavailable because of the fact that software is freely available for download on numerous ftp mirrors that are not controlled by the software producer. Even if such mirrors are controlled, not all downloads may lead to an installation.

In cases when the distribution is controlled via download tracking or licenses, there often is no way to relate outage information to the date or configuration of the system involved. Only cumulative number of systems and events are available over time. This limits the accuracy of availability estimates because the information about the age of the system when the event occurred can not be obtained. As was shown in [11], the probability of adverse events changes dramatically with time elapsed from the installation or upgrade.

Although it is possible to obtain some estimates of availability in such situations [2], we focus on the case where installation information can be obtained and linked to the outage information as discussed below. Although such rich data may not be available for many systems, it provides more accurate estimates of availability and provides basis to compare and evaluate more detailed models of availability.

The availability estimation of deployed systems is important not just in software. In fact, a routine commercial usage is most common for hardware reliability, for example, automobiles and appliances. Below, we review approaches used more broadly and compare them with the software domain. In auto industry the cost of warranty is estimated based on the repairs done under the warranty. The population of cars is obtained from the sales data and the instances of repairs from the warranty data. The age and milage of the cars determine when the manufacturer's warranty expires and, therfore, has to be available in repair instances. Repairs that are done post-warranty present an estimation challenge, because that data are not available to manufacturers. Because warranty expires based on automobile's age and milage, this presents a complex problem when estimating reliability [9, 17]. For appliances the warranty is typically provided for a fixed term. However, the repairs after the warranty period censor some observations [8]. The software producer ends up fixing the software problems and, therefore, all fixes are known to the producer. Nevertheless, the warranty affects the probability that a problem will be reported as discussed below.

Unlike cars where runtime can be approximated via milage and refrigerators where runtime is approximated via calendar time, other consumer products have more varied usage. Work in [18] considers ways to estimate actual usage time of the products. In our case of enterprise communications software the runtime is approximated via calendar time since installation, though the intensity of usage varies with daily, weekly, and other business cycles.

The ways to predict reliability early in the design cycle based on information on existing similar products is considered in [3]. For example, prediction of deployed reliability from the testing results in a large telecommunications system are presented in [19].

An excellent summary and history of availability measures for communications systems is presented in [10]. Unfortunately it does not delve into the most complex issues on how to obtain various availability measures.

Some issues of missing data in availability estimation are considered in [7], but the reference is fairly hard to obtain. A more detailed list of issues of collecting and filtering software trouble reports are presented in [5], nevertheless the issues of estimating the relevant population of systems central to our contribution are not discussed.

Although our work can be used as a basis for fitting and comparing various reliability models (see, e.g., [13]), it is not the primary objective of this study.

We continue by introducing basic information about the software system in our case study.

4. ENTERPRISE COMMUNICATIONS SOFTWARE

Our case study concerns the communications software installed on many Avaya telephony systems. This software system is an established product and embodies several decades of knowledge and experience in the telephony field. In a recent release, the software contains approximately seven million lines of code mostly in C and C++. The software development organization deploys major releases on a fixed schedule, with subsequent minor releases that bundle patches and refinements to the system.

Many releases are in the field and are used by tens of thousands of customers, many of whose businesses depend on the high availability of the product. This makes the software exceedingly difficult to enhance while maintaining the smooth operation of the hardware/software combinations deployed.

Here we are primarily concerned about information extractable from customer support and monitoring systems that track, among other things, information about the time of installation and upgrades by customers as well as customer reported outages, and automatically collected data about restarts.

5. MEASUREMENT SOURCES

In this section we describe various sources of information needed to estimate availability. We briefly present the data sources representing customer support and monitoring systems and the essential details about the way each system is operated. Although some aspects may be unique to our case study, the overall structure and process of data collection is fairly common. We have gained the knowledge of the way these support systems are operated over several years as we were investigating several questions related to customer perceived software quality. We also spent significant time validating various measures extracted from these systems. Although different from software development support systems, customer problem and inventory tracking systems have a lot in common. The largest difference is that software failures are tracked for each customer - not for each problem as happens in the bug tracking systems. The technical difficulty of linking the bug and the code in software development support systems is replaced by the difficulty of linking the configuration and upgrades of individual systems to the outages. The difficulty of recognizing if a software change fixes a bug or is an enhancement is replaced by the difficulty of recognizing if a software outage was caused by reasons not related to software.

Figure 1 shows a simplified view of operational systems and data processing presented in detail below. The primary data sources (within dotted boxes) are represented by three operational systems used to manage inventory, customer service issues, and alarms. Each one represents the lowest level in the data processing pipeline. Having several levels in the pipeline separates concerns of interfacing the operational systems (level 0), filtering, cleaning, and linking the data sources (level 1), and construction of the metrics of interest (level 2). As the operational systems (and analysis goals) evolve over time it makes it easier to maintain the entire measurement framework as each level has its own set of tests that can be run when some aspects of the data processing (or the underlying process) have changed. In addition to processing layers, the analyzed data can be always traced back to the original sources via various identifiers, such as, system, ticket, and alarm ID's, in order to validate correctness of the processing steps.

Based on the understanding of the service processes the relevant attributes are extracted from operational systems and then filtered, augmented, and analyzed as described in sections below.

In section 5.1 we repeat advantages and disadvantages of using support systems to measure and model software quality that were discussed in, for example, [11].

5.1 Advantages and pitfalls of using information from support systems

Probably the most obvious advantage of using data from support systems such as customer problem tracking system is that the data collection is non-intrusive because such systems are already deployed and used. However, that does not reduce the need for in-depth understanding of how the support systems are used.

We also benefit from a long history of past projects whose data has been captured in support systems, enabling historic comparisons, calibration, and immediate diagnosis in emergency situations. However, it takes significant time and effort to understand the data available in these systems, how to use it, and to get to the point where one can take advantage of it. Sometimes, additional data collection may be needed to facilitate modeling, as was in our case when we wanted to obtain the historic system installation dates.

The information obtained from the support systems is often fine grained, at the trouble ticket/software alarm/customer installation level. However, links to aggregate attributes, such as features and releases, is often tenuous. Furthermore, there may be challenges when cross-linking support systems in different domains, such as the alarms, outage reports, and equipment configuration.

The information tends to be complete, as every action involving development or support is recorded. However the information about what the action pertains to may be nontrivial to infer and some of the data entries, especially those not essential for the domain of activity, tend to be inconsistently or rarely supplied. Alarms may indicate system restart, but the cause may not always be clear: was the system restarted by the user or was it misconfigured at the time?

The data are uniform over time as the support systems are rarely changed because they tend to be business-critical and, therefore, difficult to change without major disruptions. That does not, however, imply that the way the systems were used was constant over the entire period one may need to analyze.

Even fairly small projects contain large volumes of information in the support systems making it possible to detect even small effects statistically. This, however, depends on the extractability of the relevant quantities. For example, missing values in several attributes may dramatically reduce the effective sample size.

The systems are used as a standard part of the project, so

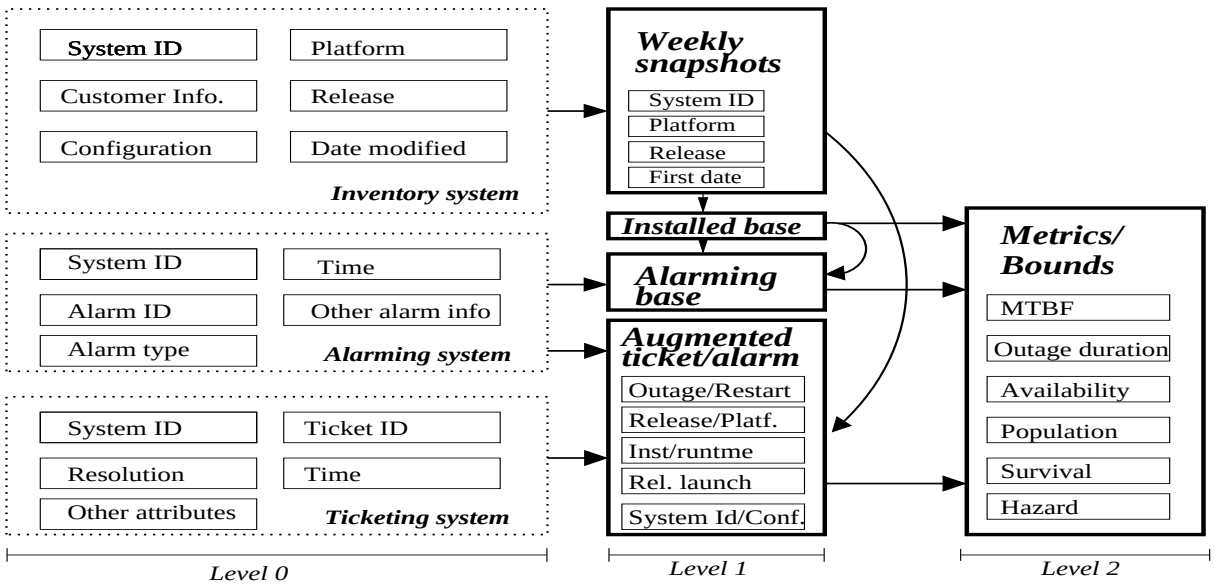

Figure 1: A simplified view of data sources and processing steps. Dotted boxes indicate operational systems and thick solid boxes represent analysis tables.

the software project is unaffected by experimenter intrusion. We should note that this is no longer true when such data are used widely in organizational measurement. Organizational measurement initiatives may impose data collection requirements that the organizations might not otherwise use and modify their behavior in order to manage the measures tracked by these initiatives.

The largest single obstacle for using the support systems for analysis is the necessity to understand the underlying process and the way the systems are used. This requires validation of the values in fields used by the developers and support technicians in order to assess the quality and usability of the attribute. Common and serious issues involve missing and, especially, default values that may render an attribute unusable. Any fields that do not have a direct role in the activities performed using the project system are highly suspect and, often provide little value in the analysis. As the systems tend to be highly focused on tracking issues or versions, extracting reliable data needed for the analysis may pose a challenge.

The following subsections detail the nature of various data sources needed to estimate system availability.

5.2 Outage information

The primary data source comes from customer support (ticketing) systems where customer reported issues are tracked until resolution. Two sources for such data are available: customer reported issues (tickets) and automatically collected alarms. Tickets represent issues where a person representing the customer has filed a report, usually via a phone call. Such issues, referred to as calls or tickets, contain information about customers' system, the description of the problem and resolution, the dates when the problem was reported and resolved, as well as a list of various teams that were involved in handling the problem. Data fields indicating standard causes and resolutions and severity are also available. The free text description contains a standard abbreviation indicating cases when a customer experienced an outage. Cause and resolution codes indicate if the problem was external (e.g. flood), if it was related to hardware, and so forth.

A large subset of systems may report issues automatically, without human involvement. We call such reports alarms. Alarms indicate the nature of the problem much more precisely. For example, alarm may indicate system or processor restart, the type of the restart (hot, warm, or cold), whether or not it was an active or a standby processor that have restarted. Many other types of alarms not related to restarts are also recorded.

Certain events such as restarts trigger additional data collection, including obtaining complete system logs over some period prior and after the restart. Such logs provide information needed to determine the cause of the event.

Each source of event data has its advantages and disadvantages. Calls tend to reflect the actual problems customers have experienced, while some alarms may be coming from misconfigured systems that are not in operational use.

However, the call data may contain human errors made while reporting the problem. If a problem is too minor to be noticed or to be reported by a customer, it may be completely missing from the call data.

Alarms provide a finer grained and more complete information about events that may cause interruptions in service, but it does not have any indication if the system was in service at the time. Certain other events may be originated by external causes, such as extended power failures or network outages, that should not be a part of software availability calculation.

To simplify our case study presentation we focus on system restarts. Although some of restarts may not cause service outages (because of the hardware duplication or a system not being in service at the time of restart) on one hand, and may not represent all service outages on the other hand, they are clearly indicating software malfunction and are responsible for a very important subset of system outages.

System restarts have several flavors. Warm restarts are the most common, last a few seconds and typically do not cause system-wide outages. Cold restarts and reloads are much more rare, but to be conservative we are focusing on the "warm" restarts. Considering all restarts provides a rough lower bound on reliability. To improve it we may consider only restarts that result in customer reported outages. That should provide an upper bound on reliability.

For the releases and products we consider in our case study there are approximately 14 thousand alarms related to software restarts. There were around 80,000 systems in the field that were running the software we study.

The next subsection describes the nature of information needed to estimate the population and runtime of the deployed systems.

5.3 Installed population of systems

Unless the software is freely distributed, the distribution and licensing are often carefully tracked. Such tracking provides the dates when systems are installed or upgraded with a new release. Often each system is uniquely identified in such an inventory and in the ticket data, making it possible to calculate system's age at the time of the problem report. In our case the calls and alarms included system ID, therefore we could calculate the age of the system at the time of the adverse event.

Unfortunately, many additional issues remain even in this case. First, not all the systems report adverse events in the same way or with the same probability. If the system is under the warranty or under the service contract, the calls do not incur costs to customer, but in the absence of service contract, calls are charged according to their severity, therefore potentially reducing the likelihood of a call. Second, some systems may not be able to report alarms either due to misconfiguration, lack of service, or customer's desire to keep their networks closed.

Therefore, for each type of adverse events, we need to estimate the population of systems that are capable of reporting them (installed base and alarming base in Figure 1) and separate them into subpopulations according to the presence of service contract and other factors that may affect the probability of adverse events being reported.

Avaya sells a range from very large to medium sized communication equipment in three hardware reliability configurations called simplex, high, and critical. Historically, the media and control information used to travel on separate communications networks, though in IP telephony that changed somewhat. Roughly, the systems with high configuration have duplicated hardware for the communication control and systems with the critical configuration have duplex hardware for communication control and for the media.

Clearly, that outage information should be treated differently for each type of configuration. For example, if a restart occurs on a standby hardware, that does not constitute an outage. We will discuss it later, but the salient point here is that we need to know the configuration of each system. Fortunately, in our case, we were able to determine the size and the reliability configuration of the installed systems, albeit not for all of them. This necessitates further subdivision of system population into the ones with known and unknown sizes and configuration.

5.4 The runtime of systems

The reliability and availability calculations require total runtime of the systems in the measured population. Once the desired population of systems is determined the runtime is determined by assuming that for each system i the system was running from time A_i to time B_i.

For a particular system i the time A_i is either the estimated installation time for the release under consideration or the time when the observation interval was started, whichever is more recent. The time B_i represents the time when the data extraction scripts are run or the time a newer release was installed on the system i, whichever is earlier.

The total runtime for a release is the sum of $B_i - A_i$ over all systems i that run or ran the release of interest. The runtime for each release/platform combination is obtained by summing over the systems representing that combination.

Although some systems do not run continously, it is a reasonable assumption that communication systems do. For other types of systems the estimate of runtime may present additional challenges. The units for runtime measures are system calendar years. It is worth mentioning, that the intensity of usage may be of interest as well. Outages during intense usage periods may bring more damage and they also may be more likely to occur when the systems are overloaded. We do not consider such phenomena here.

5.5 The duration of outages and restarts

The duration of call related outages can be estimated by calculating time elapsed between the ticket report and resolution. Unfortunately this interval does not always accurately reflect the outage duration as the ticket may not be closed immediately following the outage resolution. Sometimes it is closed days or weeks later. Therefore this duration can serve only as an upper bound on the actual outage duration.

The software restart related outages can be obtained quite precisely either from the runtime logs or from the simulations and depend on the system size, configuration and the type of the restart. The restart outage duration has three components. The first component is server related and depends on the type of the restart (warm/cold) and the speed of the processor. The second component is related to the number and type of external boards that have to be restarted in cases of a cold server restart. The last component involves the time it takes for the terminals (phones) to re-register after certain types of restarts and is primarily dependent on the number of the phones.

6. METHOD

The reliability and availability of software depends on many factors, including type, size, configuration, and usage of the system. The probability of observing an outage also depends on deployment factors, including when in the release cycle the system was deployed (early adopters tend to suffer), and the time elapsed since installation (the first weeks after installation are fraught with perils of misconfiguration). Notice, that we have not even mentioned the size and complexity of software release itself, often the only factor considered in some reliability models. This is not accidental, as other factors listed above appear to be more important in determining software availability.

Our proposal, therefore, is empirically to estimate the availability based on all the measured factors that strongly

influence it. In particular, we want to predict software availability for an individual customer. For example, we can predict the availability in the first month of operation of a large, critical configuration system, deployed six months after launch date for a particular software release. To make confidence statements about the reliability we also have to obtain the distribution of the time intervals between outages.

The inputs for such model come from the inventory and failure reports from deployed population of software systems. We describe ways to use bounds for measures that could not be accurately estimated via direct means and discuss how to overcome issues related to highly structured missing data that could dramatically bias the results.

6.1 Bounding measures of interest

Information obtained from operational systems may not always contain the most relevant measures of interest either because they were not collected, because there was some bias in the way the measures were observed, or because the desired quantity is difficult or impossible to measure exactly. One of the most difficult problems in our case was to identify outages that were caused by software and resulted in service disruption. Although apparently simple, this is virtually impossible to measure precisely. Some instances that are particularly hard to detect are when system restarts, but the restart was caused by an operator, a power failure, or by some other external circumstances. We chose to include all restarts in our estimate, even though it underestimates the actual availability.

Another kind of difficulty is to recognize restarts that do not cause service disruptions. This may be a simple case when the system is not in operational use, is being installed or (mis)configured, or a more complex case when the restart is occurring for the standby processor. A way to address this issue is to consider restarts that are associated with customer calls. This provides the upper bound on availability, as there may be service disruptions that were not noticed or not reported by the customer.

6.2 Missing data

As any real data, calls, alarms, and system inventory contain plenty of missing or misspecified values. Because the mechanism through which the data are missing vary depending on the source, we consider each separately. Although it is possible to apply various imputation techniques to repopulate some of the missing values, here we concentrate on the simplest approach in dealing with missing data — removing incomplete cases. Given the highly structured data sources, even this simplistic approach presents substantial difficulties.

6.2.1 Calls

Virtually all systems are capable of reporting calls, because the customer may call Avaya in case of any operational issues with the system. However, because the customers without a warranty or support contract have to pay for these calls, their reporting pattern is different. It is reasonable to assume that customers without the support contract are more likely to resolve issues without notifying Avaya, therefore, the probability that an outage will be reported will be lower and the availability estimates based on customer calls will be optimistic.

Another complication is that operators taking the call may not always designate it with the special code indicating an outage. The reliability would then be overestimated if the operators tend to miss including the outage designation more often than erroneously including it.

6.2.2 Alarms

Alarms, unlike calls, are automatically reported, nevertheless, the issues with the missing data remain. For example, system restarts during network outages may be not recorded. This eliminates the record for some of the outages. One can, however, argue that such outages could not have been the cause of service interruption because the network was already down.

A more complex issue arises from the fact that not all systems are capable of reporting restarts for reasons outlined earlier. If we simply obtain the total observed outage time and total runtime for all the systems, the availability would be overestimated because some of the runtime was for the systems that could not report restarts. If we are not concerned about the availability of systems that are not capable of reporting alarms or if we have good reasons to assume that their availability will be similar, we can simply investigate the population of systems that are capable of reporting alarms. That was the approach we took.

It is impossible to obtain the exact inventory of all systems capable of reporting restarts. Therefore, we use several approximations of that population. The largest set involves all systems that ever generated at least one alarm, the intermediate set involves systems that generated at least one alarm within the last 12 months, and the smallest set involves only systems that alarmed within the last six months. These bounds were based on the expert assessment that the probability a system will report at least one alarm within 12 months to be close to one.

There is, unfortunately, a potential problem with such an estimate. If the probability for a system to report an alarm is uniform over time, then the numbers for systems that have been installed for only a few months will be underestimated. Fortunately that probability is highly nonuniform over time, with initial weeks during and following installation most likely to generate alarms (see Figure 5). Although it alleviates undercounting issues for new systems, it does not eliminate it completely. Therefore, we may need to make necessary adjustments in the availability models. In particular, it is relatively straightforward to estimate the undercounting of the new alarming systems by observing the times from installation until the first alarm.

6.2.3 Outage duration

Outage duration may be especially difficult to estimate. For customer reported outages we may not have the exact outage start time as the customer may not have reported the outage immediately. We also do not have the exact time the service was restored because the customer calls are closed much later than the outages are resolved in order to investigate and eliminate potential causes of that outage. The most sensible approach to use in such situations is to assume that each outage took an average time. Such average times may be estimated for company or even industry in focused, in-depth studies.

Restart duration (indicated by alarms) can be estimated much more precisely, primarily because the time it takes to

complete a particular restart is determined by the system size, configuration, and the type of restart. Such times can be obtained in a laboratory settings or based on the system logs retrieved from the restarted systems.

6.2.4 Inventory

As any other attribute, the information on the type, size, and reliability configuration is not perfect. Some systems lack or have incorrect information about size, some about configuration, and some even about the release installed. Because misspecification appears to be rare, we focus on dealing with a large proportion of systems with missing attributes. If we can ignore the availability of systems with missing attributes of if there are reasons to assume that their availability will be the same, we can simply consider population of systems that have non-missing attributes. For availability calculations we must drop the outage time reported for systems with missing attributes. For models with a large number of predictors there may be relatively few complete cases (systems that have values for all predictors) making it difficult to obtain reliable estimates. Therefore, models with fewer predictors or models capable of handling incomplete cases may be necessary.

6.3 Other availability predictors

Even when a suitable population of alarms and systems is selected, further adjustments may be needed because the estimated availability will likely increase as we go forward from the general availability date of a software release. The estimated availability will also be likely to be negatively affected if the fraction of recently installed systems increases, or if we are considering a larger software release. We address these issues separately. First, we investigate each software release separately. To control for the system age (time since installation) and for the time elapsed since the release launch date, these factors are included in the model. Similarly, other adjustments may be taken into account by using a suitable model.

6.4 Presenting the results

In essence, our effort is concentrated on empirically obtaining three numbers: number of outages, outage duration, and total runtime. As we discussed, these three estimates should be obtained for several populations of systems and each estimate may consist of an upper and a lower bound and may include various other parameters. In this subsection lets assume that we have just one estimate for each of the three numbers and we are only concerned about what statements could be made regarding software availability.

We consider two kinds of statements that may be of interest. The first kind of statements simply considers if the observed data is compatible with the specified availability. In hypothesis testing framework our null hypothesis is that the system does have the specified availability levels and we may reject the hypothesis if the probability of observing the sample is sufficiently small. The deployed systems had specifications of three, four, and five 9's for the simplex, high, and critical configurations. The three nines availability corresponds to outage duration of .1 percent ($1-0.001 = 0.999$, hence three nines) or around nine hours per year. Four nines correspond to around one hour per year and five nines to around five minutes per year.

The other kind of statements are more conservative and consider lowest values of the underlying availability that are consistent with the observed sample. In essence, such statements would give us the lowest confidence bounds of availability that are consistent with observations. More specifically, the first kind of statements can indicate instances when availability is significantly below the specifications and the second kind of statements indicate availability levels that deployed systems are likely to exceed with high confidence.

7. RESULTS

To emphasize and illustrate the relative importance of dealing with various issues highlighted above, we present a sequence of results starting with a naive approach followed by successive refinements. If the simplest approach produces sufficiently accurate results there is no justification to spend additional effort on more sophisticated methods. It is usually better to start from the most basic approach and refine it as needed. Finally, the basic approximations may highlight artifacts or data problems that are not mentioned here, but may be salient in other projects. The pattern of successive refinements to the analysis models has a similar motivation as the division of data processing into several levels shown in Figure 1. Although data processing levels use output of the lower level as its inputs, the more sophisticated models are not built on the output of simpler models, but on data from the more refined levels.

To simplify the presentation, we focus on outages caused by restarts. Outage duration may be estimated separately and in a relatively straightforward manner as described above and is not central to our findings.

7.1 Simplest estimates

Excluding outage duration leaves us with the reliability measure or mean time between failures (MTBF). The simplest estimate of reliability is the total run time divided by the total number of outages. This "naive" approximation could be made simply from the information presented above. The total number of system restarts was approximately 14 thousand and there were about 80 thousand system installations that were running for approximately six months, giving us a rough estimate of 3 years.

To improve upon this rough estimate we will select a specific population of systems that have the simplex configuration (relationship between a restart and an outage is more complex in other configurations) and have a particular release and a particular hardware platform. We also chose a three month period (the last quarter of 2005) to reduce the effects of being close to a release date.

The simplest estimate of dividing the total runtime by the number of restarts we obtain MTBF of 8 years (we will refer to it as "Naive+" estimate). If we restrict the runtime to the estimated population of systems that are capable of generating alarms, we get MTBF of only 6 years as shown in Table 1

	Naive	Naive+	Alarming
Systems	80000	1011	761
Restarts	14000	32	32
Period	.5	.25	.25
MTBF	3	7.9	5.9

Table 1: Comparison of MTBF estimates.

The results show a substantial variation in the estimates justifying the effort spent in trying to increase accuracy. Although restricting the sample to one platform, release, configuration, and a short period of time reduces some of the variability, there are other sources of variablity, including system size and time elapsed from the installation and release launch dates. Furthermore, to make statements about the variability of the estimates, we need to understand the distribution of the times between outages. It is unlikely that the failure rate is constant over time, making the unrestricted MTBF estimate (without specifying the time period) questionable.

7.2 Restart rates

To investigate the restart occurrence in more detail we estimate how the restart rates change with time elapsed from the release installation date. We do not consider the rate for an individual system (some have a first restart early and some never restart), but rather for the entire population of systems. The difficulty of obtaining such rate empirically lies in the fact that many systems have been running for time periods that are much shorter than MTBF. Therefore, we are missing longer restart inter-arrival times. This phenomenon is often referred to as censoring and is common in medical studies, where patients leave the study before the treatment outcome is known.

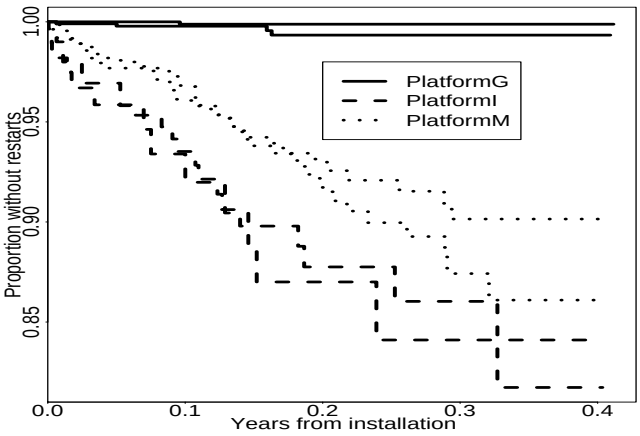

Figure 2: Kaplan-Meier estimates of the survival curves for three platforms and two releases.

Here we focus on estimating the distribution of the time until the first restart after installation or upgrade. To illustrate the similarity to clinical studies where much of the methodology to cope with censored data was developed, we provide a "mapping" between our case and terms of a simple clinical study. Each system represents a "patient" in the study. The "patient" enters the trial when the system is installed or upgraded to a new release and leaves the trial when the first restart occurs or when the next upgrade to a new release is done. The "trial" ends at the time the analysis is performed. The presence of a restart indicates the negative outcome (patient dies). In case of no restarts the outcome is positive (patient survives).

A common statistical technique to deal with censoring is to estimate a survival curve using Kaplan-Meier Estimate [6, 4]. The survival curve is a graph showing the percentage surviving versus time. The survival curve is also known as the reliability function in the context of hardware or software. We can use the curve to estimate the distribution of restart times, reliability, or other quality characteristics. Knowing the distribution of restart times would also allow making statements about the confidence intervals for the reliability estimates.

For the survival curve the vertical axis gives the proportion of systems without restarts (surviving). The horizontal axis gives the time elapsed after the system installation. Figure 2 illustrates Kaplan-Meier estimates for two releases and three platforms installed or upgraded since October, 2005. There is a large variation among platforms — around one, ten, and fifteen percent have a first restart within four months (0.3 years). The survival curve for the second release is below the one for the first release for all three platforms. This indicates that time elapsed from the release launch date (second release was more recent) and/or release size (the second release was larger) may have reduced reliability. The plot was generated using R [16] package *survival* [15]

In order to investigate if the probability of failure is constant over time we present the estimate of the hazard function in Figure 3 for the same three platforms (we have not separated the releases when estimating the hazard function). Hazard function is an instantaneous probability that a system will restart at a specified time after installation provided it has not restarted before that. Figure 3 shows estimates based on the work in [12] as implemented in the R package *muhaz* [14].

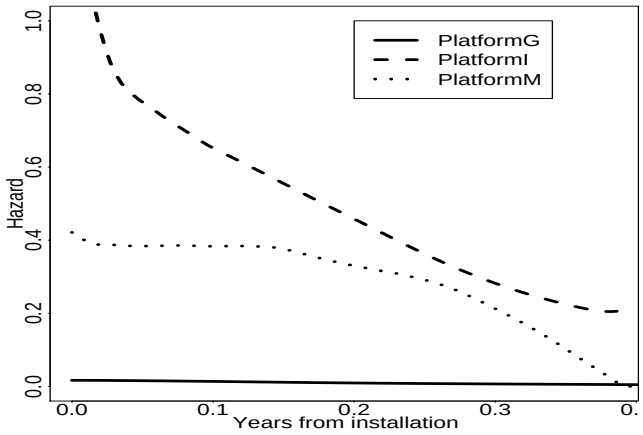

Figure 3: Hazard function for three platforms.

The hazard function appears to be decreasing over time (at least for the considered time interval) and varies among platforms. Figure 4 shows the hazard function for the platform G that appears to be flat in Figure 3. It follows a similar pattern to other platforms.

7.3 Adjusting population counts

As described in Section 6.2.2 the population of alarming systems is not known exactly and has to be estimated. The estimate considers systems that previously had alarms. Although alarms happen quite frequently (alarms are generated for many reasons, not just as a result of the infrequent occurrences of software restarts that are the subject of this investigation), it is clear that the population of newly installed systems will be underestimated. The extent of the underestimation is illustrated in Figure 5.

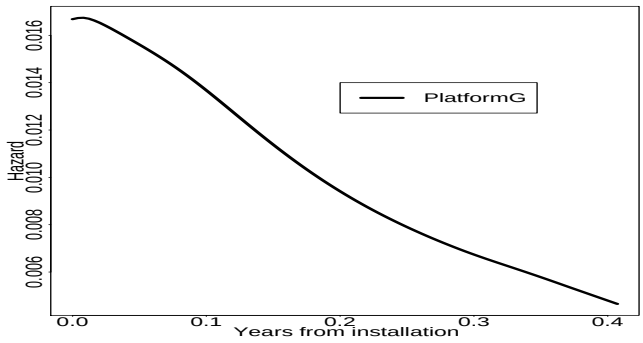

Figure 4: Hazard function for platform G.

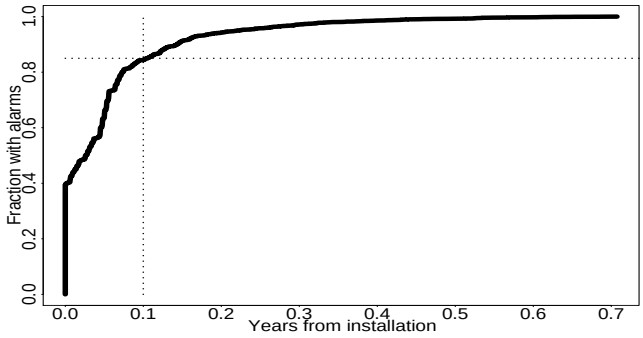

Figure 5: Distribution of time until first alarm. 85% of the systems generate an alarm within 0.1 years of operation.

As it is possible to assess the extent of underestimation it is also possible to adjust the survival and hazard estimates. For example, we can generate additional observations for each system that did not have a restart (because restarts are instances of alarms, a system with a restart can not be missing from the alarming population). The number of additional observations would be proportional to the age (runtime) of the alarming system. For example, for each system that ran 0.1 years we generate $1/0.85 - 1$ additional observations. Because we can not add fractional observations we can generate one with that probability. For comparison, unadjusted and adjusted hazard estimates are presented in Figure 6. Because the adjusted estimates are based on a larger system population, the hazard is lower, especially for the newly installed systems.

8. DISCUSSION

The investigation suggests a strong effect various attributes may have on software reliability and availability in the considered software and illustrates that the failure rate may not be constant over time, across platforms, or releases. This makes it difficult to make any general statements about system MTBF or availability and suggests that the simplest estimation methods may provide inaccurate picture. For example, a customer may want to know the availability of a new system they are considering. The availability would depend on a number of factors including the system size and configuration and on dates the release was launched and installed. Another practical question may arise if a software provider would like to assess if the reliability of a new release

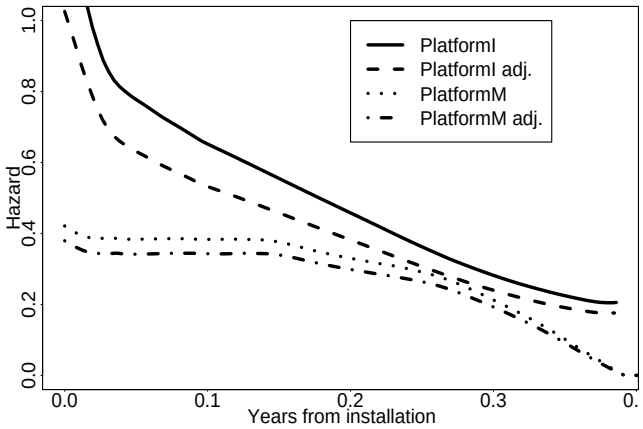

Figure 6: Comparison of unadjusted and adjusted hazard functions.

has improved in comparison to a previous release. Unless the estimation models include the key attributes known to affect the reliability, any differences between the empirical estimates for two releases are likely to be driven by the differences between populations of systems running two releases. For example, if a new release is more frequently deployed on larger systems the size-unadjusted estimate is likely to show lower reliability even if size-adjusted estimates are the same.

However, we can still discuss reliability and availability for a selected platform and for a specific interval of time. The estimated hazard and survival functions provide the basis to make confidence statements about the empirical estimates of reliability. We may also investigate the most appropriate distributions for the restart arrival times and build more complicated models. Such investigations are beyond the scope of this paper.

Given that the MTBF may be difficult to interpret unless we fix a specific time period, we are left with the question of what are the most suitable measures of software reliability when the failure rate is not constant? The answer may depend on the type of software system, but for long-running communication systems it may make sense to look for a probability that a system restarts before the next upgrade. In the domain we are investigating it is not unusual to see the upgrades occur about every two years. Similarly, availability could be defined as the fraction of time the system is operating within the first two years of installation. Obviously, the appropriate interval may be different for other types of software that has shorter (or longer) runtimes before an upgrade. The probability of a restart within 5 months can be read directly from Figure 2 and varies from less than one percent to more than fifteen percent depending on release and platform.

Despite their drawbacks, the simplest models have a significant role in the estimation process. Given the complex structure of data sources the estimation tools are a relatively complex software system in themselves. The simplest models serve the role of regression tests that can provide an immediate feedback if some aspects of that analysis system become incorrect either because of changes in the analysis system, in the operation support systems, or in the business process. Similarly, the separation of data processing into

several levels helps to maintain and adapt the analysis system to changes brought by evolution in the tools, processes, and analysis requirements.

9. SUMMARY

We have investigated empirical estimation of reliability and availability of deployed systems in the context of communications software. The proper estimates are difficult to obtain because of difficulty of measuring relevant quantities and complex structure of missing data.

We suggested using approximations to bound the desired estimates and described various pitfalls if the adjustments are not made for the missing data. We propose several ways to deal with missing data and obtain relevant populations when estimating system reliability and availability.

We looked into the practical question of whether an easy to compute approximations would be accurate and found them to be substantially different from the more accurate estimates that take into account the way the data was collected.

We also evaluate if the common measure of the communication systems reliability in terms of mean time between failures (MTBF) is meaningful for varying runtimes by investigating the hazard function. Our findings indicate that software reliability strongly depends on several factors, and the failure rate is not constant over time. This suggests that the traditional measures that imply constant failure rate may have to be interpreted with caution.

10. ACKNOWLEDGMENTS

We would like to thank Bahareh Momken, Luke Young, and many other experts that provided invaluable help and suggestions for this work.

11. REFERENCES

[1] D. Coit and W. Turkowski. Practical reliability data and analysis. *Reliability Engineering*, 14(1):1–17, 1986.

[2] D. W. Coit and K. A. Dey. Analysis of grouped data from field-failure reporting systems. *Reliability Engineering and System Safety*, 65(2):95–101, 1999.

[3] J. Fahy. Estimating warranty and service costs from mtbf estimates. In *Electro/95 International. Professional Program Proceedings*, pages 35 – 47, 21-23 June 1995.

[4] T. H. Fleming and D. Harrington. Nonparametric estimation of the survival distribution in censored data. *Comm. in Statistics*, 13:2469–86, 1984.

[5] K. Kanoun, M. Kaaniche, and J. Laprie. Experience in software reliability: From data collection to quantitative evaluation. In *Proceedings, Fourth International Symposium on Software Reliability Engineering,*, pages 234 –245, 3-6 Nov 1993.

[6] E. Kaplan and P. Meyer. Non-paramentric estimation from incomplete observations. *J Am Stat Assoc*, pages 457–481, 1958.

[7] C. Kjaergaard. Field failure data collection and analysis of repairable systems. In *Reliability Data Collection and Use in Risk and Availability Assessment. Proceedings of the 6th EuReDatA Conference*, pages 848–858, 1989.

[8] K. Liu. Refrigerator failure early prediction based on warranty data. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 195–199, 2002.

[9] M.-W. Lu. Automotive reliability prediction based on early field failure warranty data. *Quality and Reliability Engineering International*, 14(2):103–108, March-April 1998.

[10] H. Malec. Communications reliability: a historical perspective. *IEEE Transactions on Reliability*, 47(3):333–345, Sept. 1998.

[11] A. Mockus, P. Zhang, and P. Li. Drivers for customer perceived software quality. In *ICSE 2005*, St Louis, Missouri, May 2005. ACM Press.

[12] H. Mueller and J. Wang. Hazard rates estimation under random censoring with varying kernels and bandwidths. *Biometrics*, 50:61–76, March 1994.

[13] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability*. McGraw-Hill Publishing Co., 1990.

[14] S. original by Kenneth Hess and R. port by R. Gentleman. *muhaz: Hazard Function Estimation in Survival Analysis*. R package version 1.2.2.

[15] S. original by Terry Therneau and ported by Thomas Lumley. *survival: Survival analysis, including penalised likelihood*. R package version 2.20.

[16] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.

[17] B. Rai and N. Singh. Hazard rate estimation from incomplete and unclean warranty data. *Reliability Engineering and System Safety*, 81(1):79–92, July 2003.

[18] N. Sarawgi. A simple method for predicting the cumulative failures of consumer products during the warranty period. In *Annual Reliability and Maintainability Symposium 1995 Proceedings*, pages 384–390, 1995.

[19] X. Zhang and H. Pham. Predicting operational software availability and its applications to telecommunication systems. *International Journal of Systems Science*, 33(11):923–930, Sep 15 2002.