

# Drivers for Customer Perceived Software Quality

Audris Mockus and Ping Zhang

Avaya Research

233 Mt Airy Rd

Basking Ridge, NJ 07920

audris@mockus.org,pingzhang@avaya.com

Paul Luo Li

Institute for Software Research International

School of Computer Science

Carnegie Mellon University

Pittsburgh PA, 15213

Paul.Li@cs.cmu.edu

## Abstract

*Predicting software quality as perceived by a customer may allow an organization to adjust deployment to meet the quality expectations of its customers, to allocate the appropriate amount of maintenance resources, and to help direct quality improvement efforts to maximize return on investment. However, customer perceived quality may be affected not simply by the software content and the development process, but also by a number of other factors including deployment issues, amount of usage, software platforms and hardware configurations. We predict customer perceived quality as measured by various service interactions, including soft-ware defect reports, request for assistance, and field technician dispatches using the afore mentioned and other factors for a large software system. We employ the non-intrusive data gathering technique of using existing data captured in automated project monitoring and tracking systems as well as customer support and tracking systems. We find that the effect of deployment schedule, hardware platform, and soft-ware configurations can increase the probability of observing failures more than 20 times. Furthermore, we found that the factors affected all quality measures in similar fashion. Our theoretical model could be applied at other organiza-tions, and we suggest methods to independently validate and replicate our results.*

## 1. Introduction

Anticipating customers experience with a new software release has significant business importance. The decision to release software early may be necessary due to competi-tive pressures and the need for additional revenue. How-ever, a product that does not satisfy customers' quality needs may incur additional expenses in repair, maintenance, and future business opportunities. Our main focus is to predict customers' experiences within three months of deployment to facilitate resource planning, software deployment, and

other key aspects of the software business. We use customer perceived quality, and customer experience interchangeably, even though each term has a slightly different meaning.

We are also interested in quantifying the relative impor-tance of various process and product factors on customer ex-perience, which can help guide quality improvement to max-imize the return on investment.

We examine deployment issues, usage patterns, cus-tomers' software platform, and customers' hardware config-urations.

Finally, we want a method that can be easily adapted and used at many organizations. Therefore, we attempted to use data that is available at other comparable organizations. We describe our data collection and data analysis methods so our experiments can be replicated and independently validated.

Our approach is first to design and operationalize a small set of customer experience measures and product/process factors, then create and validate a customer experience model based on these measures, and finally to produce predictions that answer practical questions essential to software business. Our primary motivation was to answer two basic questions:

1. What is the likely customer experience for a particular customer?
2. What resources will be needed to handle the flow of cus-tomer reported issues?

We use measures and factors gathered during the customer support process from a variety of systems used to track cus-tomer installations, updates, and complaints. We also use measures and factors gathered during development from in-formation in various change management systems used to track and manage software development activities.

To operationalize customer perceived quality measures we spent significant amount of effort to familiarize ourselves with the business processes used in customer support and product development, while paying particular attention to the usage of tools that support these processes. We applied var-

ious validation techniques to ensure the accuracy of the extracted measures.

We find predictors relating to deployment date, software platform, upgrades, and system size to be significant in determining customer experience and affecting release planning. The models quantify the relative importance of each factor and enable informed decisions regarding deployment strategy and staffing needs. The predictive value of the models varies based on the frequency of the predicted event: rare events like a customer reported problem that cause a change in software are more difficult to predict than frequent aggregated events like monthly customer calls.

We start by describing our motivation in Section 1.1. The background on projects under study and the analysis methods used is in Section 2. Section 3 introduces our models of customer experience and Section 4 presents our findings. Section 5 discusses the validity of the results and section 6 outlines how similar prediction techniques could be applied in other projects. We conclude with related work in Section 7 and discussion in Section 8.

## 1.1. Motivation

While there are techniques to determine the remaining faults in unchanging software (see, e.g., [15, 7]), predict which modules (see, e.g., [8]) or changes (see, e.g., [13]) will have defects, how much effort the defects repair will require (see, e.g., [1]), the release of a software system is typically full of unknowns: when the quality is finally good enough, what a particular customer will experience, how much resources needs to be devoted to support and to current engineering, etc. Here we focus on customer experience prediction models to quantify various factors that affect a range of customer experiences and provide a quantitative basis for making business decisions.

Such findings can enable a company to assess the quality being delivered to a specific customer and allow for deployment or other adjustments that ensure the quality expectations of the customer are met. Knowing the extent to which each factor influence customer perceived quality can also allow for targeted improvements in development and support processes or to the software itself, that can maximize the return on investment. Finally, we consider addressing resource planning issues. Since we model and predict customer reported problems, dispatches, and field software defects, we can aggregate them to predict customer support and development resources that will be needed.

The end users of a software typically experience the quality of the entire “solution” that includes the physical system, terminals, and networks, as well as the “pure” software platform including operating systems, servers, clients and other software components that are required to use a particular piece of software. Therefore, it is often difficult to separate failures in the surrounding hardware, network, and software environment from the failures of the software under study.

Human factor problems such as software upgrade and configuration often yield their own significant share of failures. Furthermore, determining the quality that will be delivered by the products is a complicated task, given the research indicating that customer satisfaction with a product is far more complex than merely knowing the number of latent defects in a system [4].

This research seeks to capture and predict some measurable aspects related to customer perceived quality in a quantitative model that could explain and order various factors in terms of their effect on issues perceived by a customer.

To determine the various measures and factors of customer perceived quality we avoid creating novel measurements but, rather, we rely on the process, tools, and information that established companies with a large customer base already use to provide support. While this information may not contain everything that affects the customer, it is sufficient for established companies to stay in business, and therefore must contain the essential insights relevant to retaining and satisfying customer needs.

## 2. Background

In this section we describe the context where we applied our model. We describe the software project, the system that tracks development, and the systems that track and monitor deployed products. We also briefly describe the methodology we used to extract data from the tracking databases.

### 2.1. The software project

We examine the call processing 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++ languages. The system deploys major releases on a fixed schedule, with subsequent dot releases that bundle patches to the system and incorporate refinements.

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

We use customer interaction measures and quality factors captured in two types of systems. One type of systems contains service request information (e.g., trouble tickets) captured in the post-sale customer support process. The other type of systems contains information captured during product development in the software change process. We use the product development system to identify customer issues that lead to software changes.

The project under consideration used the Sablime system for problem tracking and an internal system for most of the

version control. The modifications done as a result of customer interactions described below can be traced back to the related interaction. More details on the nature and analysis of software change data is provided in, for example, [14].

### 2.1.1 The customer support process

Avaya uses a tiered support process similar to those in other organizations [18, 4], and has smart agents installed on deployed products.

The data for our analysis come from two sources: The trouble ticket database and the equipment database. A trouble ticket is generated for each customer contact, whether it be an alarm sent by the smart agent or a phone call from a real person. The ticket contains the usual information that allows a problem report to be associated with a customer and an installed product. A ticket can be routed, escalated, or dispatched depending on the type of the problem and the type of service agreement the customer has. There were over 4 million tickets created in 2003 and roughly half are related to the products we analyze in this paper.

The equipment database contains information about products installed in each customer location, regardless of whether the product has reported any trouble ticket. Typical product attributes include software release, number of licensed ports, and product configuration, etc.. This information is updated whenever a change in these attributes occurs (e.g., software updates). There are over 4 million locations listed in the equipment database, although only around 100K locations contain products that we consider.

## 3. Models of customer perceived failures

We assume that customers' perception of quality will be negatively affected if customers are distracted away from their normal business activities by problems with the system. We distinguish two major aspects of customer perceived quality:

- impact of problem occurrence
- frequency of problem occurrence

In addition we assume that these problems are related to how the software is deployed, operated, and configured. Here we focus on aspects of customer experiences that are captured during the course of product deployment and maintenance, and ignore other important aspects like price, feature richness that may be collected as a part of marketing efforts.

We are interested in rare high impact problems, which are problems that do not occur frequently but which are costly in terms of time and resources needed to resolve the problems and the interruption to the customers' business. A defect occurrence that leads to a change in code is an example of this type of problem. It is costly to the customer because the problem needs to be escalated through the tiered customer

support organization which takes time. The problem would remain unresolved over long duration which may reduce the functionality of the system for the customer.

We are also interested in frequent low impact problems, which are problems that do not have a high per incident impact, but which, if occur frequently, would negatively impact customer perceived quality. A customer call into the support center is an example of this class of problems. A single call might not be a major interruption to the customer. However, if the customer needs to make hundreds of calls within the first three month then it might become a serious issue.

In the following sections we define and operationalize measures for customer perceived quality and the predictors based on information captured in the project monitoring and customer support systems. Similar data is available for most high availability business-critical software products and is discussed in Section 7.

### 3.1. Measures of customer perceived quality

We consider measures that can be extracted from Avaya's customer support systems that have been developed and improved over several decades. They are also similar to measures reported to be significant in other research works [4].

- Rare high-impact problems
  - equipment service outages
  - malfunctions resulting in software modifications
- Frequent low impact problems
  - technician dispatches
  - customer calls
  - alarm reports

We call these measures of customer perceived quality *customer interactions*. Each of these customer interactions approximates slightly different aspect of the customer experience. The measures reflect experiences that vary in severity and amount of time until resolution. For example, issues caused by alarms can often be automatically fixed without human intervention in a very short period of time, while issues requiring software modifications can take several weeks to diagnose and resolve via delivery of a patch.

We have chosen to model the occurrence of the measures in the first three months from the deployment, because this is a reasonable time to set up, configure, and tune even a very sophisticated software system such as switching software. It also appears that the initial period is most fraught with risks, therefore of most concern to the software provider and to the customer. For other products a different interval may be needed.

## 3.2. Predictors of customer perceived quality

We examine deployment issues, usage patterns, customers' software platform, and customers' hardware configurations. Ideally, we would like to have orthogonal measures of the factors so we can isolate the effects of each factor, however such perfect measures are rarely available in empirical studies. The operationalized measures of the factors, which we call *predictors*, sometimes measure several factors.

For example, one of the predictors is the number of ports on an installed system, which measures the usage factor, the hardware configuration, and the software configuration factor (since additional software and hardware components are usually associated with larger systems). Some predictors overlap and measure the same factor. For example, the predictor indicating which type of hardware is installed overlaps with the number of ports as measures of the hardware factor. We discuss this further in Section 4.

### 3.2.1 System size

The system size predictor measures the hardware configuration factor, software platform factor, as well as the usage pattern factor. We consider systems running on small to medium and large platforms. The deployed hardware have different capabilities and larger systems have hardware and software components to support special functions or devices that are not present in smaller systems. In our models, we encode this an indicator variable called LARGE.

We expect smaller systems to generate fewer customer perceived quality issues. First, there are fewer things to configure and fewer other systems to interface with. Second, smaller systems may not have as much usage as larger systems. Finally, smaller systems may not be as likely to be involved in business critical applications that requires 7x24 uptime. Consequently, customers of such systems are less likely to experience and report issues.

### 3.2.2 Operating system

The operating system predictor measures the software configuration factor. We consider systems running on small, medium, and large platforms under a proprietary and an open (Linux) operating systems. A very small version of the system is also available on WindowsNT/Windows2000 platform. This is a categorical variable with three levels. In our models, we encode it using two indicator variables, OX and WIN.

We expect off-the-shelf operating systems to introduce unnecessary complexity and configuration issues that can be more easily controlled in a proprietary system (OX), where only essential features are supported and versions and configurations can be precisely controlled to reflect environments used in system verification. We do expect that more complex operating systems (WIN) would introduce more issues.

### 3.2.3 Ports

The ports predictor measures the usage pattern factor and the hardware configuration factor. The number of ports indicates how many licensed endpoints are supported by the system. The usage patterns would likely to be very different for machines with different number of ports. A switch supporting 3000 users is likely to be operated closer to performance limitations than a switch supporting 100 users and the intensely used switch that reaches the performance limits may be more likely to experience problems. In our models, we encode the log number of ports with the  $\log(nPort)$  variable.

We expect systems with more ports to be more likely to report a customer interaction. This is due to both the increased amount of usage and increased usage at borderline and complex situations.

### 3.2.4 Total deployment time

The deployment time predictor measures the deployment issues factor. We use total system runtime on all deployed systems from the deployment of the first system until the deployment of the system  $j$  as a measure of deployment time:  $Runtime(t_j) = \sum_{t_i < t_j} (t_j - t_i)$  where  $t_i$  is the deployment times of the  $i$ -th systems. In our models, we encode log of the total deployment time with the  $\log(runtime)$  variable.

As the system with the new release is exposed to customers and their varied usage patterns, more software issues may be exposed. These issues are then fixed via patches and incorporated in the later dot releases that are delivered to customers. Because these dot releases would be shipped later in the major release deployment cycle, they would have resolved the issues detected by early customers and should demonstrate better quality. As we gather experience in the detection and remediation of problems, as we improve product knowledge through installation and configuration, the technicians and customers will become better in avoiding future problems. Therefore we should expect fewer software problems as the total deployment time increases.

### 3.2.5 Software upgrades

The software upgrade predictor measures the deployment issues factor. Upgrades (indicator variable Upgr) are specific cases where the software received an upgrade within three months prior to installation of a major release.

In general, upgrades serve to keep machines running properly by incorporating the latest fixes and refinements to the system. Upgrades have the clearly defined purpose of making the system more stable, so we expect them to have that effect.

## 3.3. Nuisance factors

In addition to product and process related metrics, it is clear that customer reporting practices and organizational

factors also affect customer interactions. This may include industry segmentation (financial, government, health care, etc), customer support organization (US domestic vs international), or company size (large cap, mid-cap, etc). In this analysis, we picked variables that we think should have significant effect. We call them nuisance factors because they are likely to identify peculiarities of data reporting and collection, but not necessarily the differences in the underlying customer perceived quality.

### 3.3.1 US or international installation

We make a distinction between US domestic and international customers largely because the support processes differ significantly. There might be other differences, for example, the system may interfaces to a slightly different equipment and networks. In our models, we encode this using the US variable.

### 3.3.2 Service contracts

We make a distinction between customers that have and do not have service contracts. If a customer does not have a service agreement, then each time it requests for help, the cost will depend on the nature of the problem, plus the appropriate parts and labor charges. We speculate that customers without a service agreement tend to report issues less frequently and, when reporting, less likely to indicate it as a high severity issue that would require more urgent and more extensive intervention. By contrast, customers under full coverage support tend to call in more often and tend to inflate the severity of an issue to get more immediate attention. In our models, we encode this using the Svc variable.

In addition, customers with an Avaya service agreement also get remote monitoring service that may prevent some of the issues from having significant impact by implementing quick (and automatic) fixes.

We use a service contract variable in the model. This measure is likely to be confounded with customer types. Customers that require very high availability are more likely to pay for a full coverage service agreement.

### 3.3.3 Missing configuration information

The deployment data, especially the number of ports variable, has a large number of missing entries. The proportion of missing data for the number of port is large (44%) and the customer population where data is missing may be different, making conventional statistical treatment of missing data (e.g. imputation) inappropriate. We introduce additional indicator variable nPortNA in the analysis to identify the systems with missing value for the number of ports.

## 4. Results

In this section, we present results of regression analysis. The response variables are all in the form of event counts except for the very rare event of software malfunction which we convert to a binary indicator (i.e., whether the count is positive or not) and fit a logistic regression model. For other measures we take a log transformation of the response variable and fit a linear regression model [17].

We first fit a model to test the relationships hypothesized in Section 3 using the data from a single major release. Then we use that model to predict customer interactions for the next major release.

Due to space limitations, we present full results only for two measures that reflect the two types of quality issues of interest described in section 3. We briefly discuss the results for other measures later.

### 4.1. Software failures

We attempt to predict if a customer will observe a failure that will lead to a software change using logistic regression. Such events indicate severe problems that can not be fixed in a short time. It also puts strain on development resources. The rate of such problems needs to be carefully predicted and controlled so that they do not exceed available support and development resources.

Our response variable  $Y_i^{MR}$  is binary. One, if a customer reports a problem leading to a software modification within the first three months after deployment and zero otherwise. Our predictor variables,  $\tilde{x}_i$  are described in Section 3.2. Failures that lead to a software change is a rare event and a common technique to model this binary responses is logistic regression:

$$\mathbb{P}(Y_i^{MR} = 1 | \tilde{x}_i) = \frac{e^{\tilde{x}_i^T \beta}}{1 + e^{\tilde{x}_i^T \beta}}$$

#### 4.1.1 Modeling software failures

	Estimate	Std. Err.	z-value	Pr(> z )
(Intercept)	-5.26	0.64	-8.18	$3 * 10^{-16}$
$\log(rttime)$	-0.30	0.03	-8.85	$< 2 * 10^{-16}$
Upgr	1.38	0.15	9.01	$< 2 * 10^{-16}$
OX	-1.18	0.17	-6.75	$2 * 10^{-11}$
WIN	1.01	0.34	2.98	0.003
$\log(nPort)$	0.36	0.08	4.37	$10^{-5}$
$nPortNA$	2.03	0.58	3.49	$5 * 10^{-4}$
LARGE	0.52	0.20	2.67	0.01
Svc	0.57	0.18	3.11	.002
US	0.52	0.27	1.92	0.05

**Table 1. Software failure regression results.**

The results are presented in Table 1. Total deployment time (*rtime*), as described in Section 3.2, decreases the likelihood that a customer will observe a failure leading to a software change. Existence of upgrades (*Upgr*) increases the likelihood of failure. The proprietary operating system (*OX*) decreases, and the “Windows” platform (*WIN*) increases the likelihood of failure. The likelihood of failure increases with the number of ports (*nPort*) and also in cases where the number of ports is not reported (*nPortNA*). Very large systems (*LARGE*) are more likely to experience failures. Finally, the two nuisance factors indicate that customers with service contracts (*Svc*) and in the United States (*US*) are more likely to report software failures.

The model reduces the deviance by about 400, the residual deviance is still quite high: around 2000, indicating that there is still a lot of unexplained variation. This is not particularly surprising because software development, verification, deployment and service processes are designed to eliminate all failures. Consequently, if there was a clear pattern explaining when failures occur, the relevant organizations would have taken measures to address these issues and that would have lead to a more random pattern.

The model indicates that the total system run time is one of the most important predictors of failures. It is important to understand why such difference exists: the customers who installed the application earlier may have detected malfunctions that are fixed by the time later customers install their systems. Also, the individuals performing installation and configuration increased their experience themselves or through improved documentation (by documentation we also have in mind emails, informal conversations, and discussion lists) and training increasing awareness of potential problems and work-arounds.

The lesson from this relationship is that customers with less tolerance to availability problems should not be the first to deploy a major software release. Obviously this is well known practice that is often expressed as a qualitative statement “never upgrade to dot 0 release.” Depending on system configuration the probability of failure drops from 13 to 25 times for the most reliable proprietary operating system as runtime goes from the first system installed to the system that has midpoint in terms of *rtime* predictor. The least reliable windows platform experiences drop between 4 and 8 times, and for Linux platform probability drops between 7 and 24 times depending on configuration. This indicates that for the most reliable platforms the deployment schedule has tremendous impact on customer experiencing software failure.

The number of ports provides significant input even after adjusting for the system size. Since the number of licensed ports also represents system utilization, we may infer that utilization is also important in predicting the reports leading to changes as hypothesized.

The only surprise is that the upgrade indicator is related

to a higher probability of failure. This suggests that upgrades may be manifestation of the complexity of the systems that increases the likelihood of upgrades and failures.

Nuisance parameters require slightly different interpretation because they distinguish among the populations of customers and the different reporting processes. Consequently, they reflect differences in data reporting and collection. The positive coefficient in Table 1 for the *Svc* variable may appear to be counterintuitive because having a service agreement should help reduce error rate, i.e., a negative coefficient for *Svc*. Our interpretation, for more detail see [19], is that having a service agreement significantly increases a customer’s willingness to report minor problems, so much so that we are not making an apples-to-apples comparison. To really measure the effect of service agreement, one should control for the over-reporting effect by looking at customers with similar experiences. This is known as a case control study, see, e.g., [3], in the statistical literature.

#### 4.1.2 Predicting software failures

To have results applicable in practice we predict software failures for a new release based on our fitted using previous releases (in this case one previous release). We refitted the model in Table 1 to perform the prediction. We used only the most significant predictors that had p-values below 0.01 as shown in Table 2. We also excluded the predictors  $\log(nPorts)$  and *nPortNA* because those predictors were not available to us at the time of analysis.

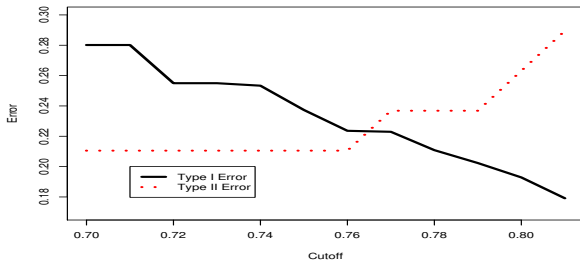
	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	-2.58	0.27	-9.64	$< 10^{-16}$
$\log(rtime)$	-0.30	0.03	-9.10	$< 10^{-16}$
Upgr	1.69	0.15	11.51	$< 10^{-16}$
OX	-1.34	0.16	-8.64	$3 * 10^{-12}$
WIN	0.61	0.30	2.01	0.04
Svc	1.03	0.15	6.67	$3 * 10^{-11}$

**Table 2. Software failure prediction model.**

We used the parameter values in Table 2 and data from the new release to predict the likelihood of software failure for the new release. The customers that have predicted probability of failure above a certain cutoff value *c* are predicted to experience such failure. These predictions are then evaluated by comparing them to what actually happened. The characterization of the prediction is based on Type I (the proportion of systems that do not fail but that are predicted to fail) and Type II (the proportion of systems that do fail but that are not predicted to fail) errors.

The plots of the two types of errors for different cutoff values are in Figure 1.

The horizontal axis in Figure 1 shows the cutoff in terms of the quantile of the probability (fraction of customers that



**Figure 1. Type I and II errors.**

have probability below certain value) rather than actual probability for confidentiality reasons. All predicted probabilities are less than 3% indicating that failure is a rare event.

To choose an appropriate cutoff value, we need to look at error probabilities for a range of cutoff values and to conduct a cost-benefit analysis. The decision may be different for different products and customers. To satisfy customers least tolerant to failures higher cutoff value should be used than for customers that are more aggressive exploring new capabilities.

## 4.2. Customer calls and other quality measures

We analyze the number of calls, system outages, technician dispatches, and alarms within the first three months using linear regression. For example, in the case of calls, the response variable  $Y^{calls}$  is the number of calls in the first three months transformed using the log function to make errors more normally distributed. The predictor variables,  $\tilde{x}_i$  are described in detail in section 3.2. Here is the model:

$$\mathbb{E}(\log(Y_i^{calls})) = \tilde{x}_i^T \beta$$

### 4.2.1 Modeling customer calls

	Estimate	Std. Err.	t value	Pr(> t )
(Intercept)	0.35	0.04	7.90	$3 * 10^{-15}$
$\log(rttime)$	-0.08	0.00	-27.72	$< 2 * 10^{-16}$
Upgr	0.73	0.02	46.78	$< 2 * 10^{-16}$
OX	0.13	0.01	9.62	$< 2 * 10^{-16}$
WIN	0.75	0.03	25.73	$< 2 * 10^{-16}$
$\log(nPort)$	0.10	0.01	16.82	$< 2 * 10^{-16}$
nPortNA	0.39	0.04	10.80	$< 2 * 10^{-16}$
LARGE	0.30	0.01	20.78	$< 2 * 10^{-16}$
Svc	0.28	0.01	23.06	$< 2 * 10^{-16}$
US	0.41	0.01	28.99	$< 2 * 10^{-16}$

**Table 3. Number of calls regression.  $R^2 = .36$ .**

Most predictors have high statistical significance due to large sample sizes that we observe. Table 3 shows the fit-

ted coefficients. The regression results for the remaining 3 quality measures (system outages  $R^2 = .06$ , technician dispatches  $R^2 = .15$ , and alarms  $R^2 = .18$ ) are all pointing in the same direction except as noted below.

The runtime factor improves (decreases) all five quality measures and is highly significant as in Tables 3 and 1. Existence of upgrades (Upgr) makes all five quality measures worse. As we discussed above, it is likely to capture systems that are more complex in ways that other measured factors can not do. Larger (LARGE) systems fare worse than small and medium systems across all measures. Within small and large systems the increase in the number of ports (nPorts) decreases quality with respect to all five measures indicating that higher usage associated with larger number of ports increases the likelihood to have negative experiences.

The complexity of the operating system did not always have hypothesized effect. The numbers of outages, dispatches, and calls were lower for Linux than for the embedded system. We do not have a good explanation of this discrepancy. Furthermore, the Windows platform, as hypothesized, had all the quality measures significantly worse (at .001 level) than other platforms except for the number of alarms. This is not particularly surprising since only a few types of alarms are generated on these very low end systems.

Despite the few exceptions, it is reassuring to see that such diverse measures of customer perceived quality appear to be affected in almost the same fashion. It implies that, at least in terms of studied measures, the aspects of quality do not need to be traded against each other. Simultaneous improvements are possible.

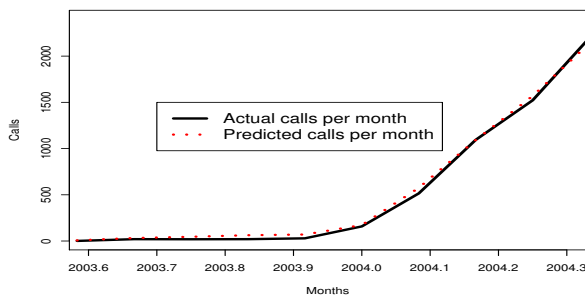
### 4.2.2 Predicting customer call traffic

Here we illustrate how to predict customer call traffic for all new customers to determine staffing needs of a customer support organization. On average, each customer support specialist can process a fixed number of calls per month, so to predict staffing it is sufficient to predict the total number of calls per month. We use the model as in Table 3 but without nPorts variable to predict the number of calls for a new release. Figure 2 illustrates the trend of predicted and actual inflow of calls for new systems.

The two trends are very close to each other indicating that the total flow of calls can be predicted fairly accurately. More practical prediction should include inflow of calls for new and existing systems, however due to space limitations we do not present full details of such prediction.

## 5. Validation

It is important to validate data, measures, and models to make sure that trends reflect underlying phenomena and not the peculiarities of the data collection method or specifics of a particular project.



**Figure 2. Prediction of monthly call traffic.**

We first inspected documents related to the development and support process and interviewed relevant process experts to verify their accuracy. Through this process, we discovered differences between different populations of customers, which lead to the inclusion of location and service predictors into the models.

External validation involved interviewing experts in the field personnel to make sure that results correspond to their perception of the reality. We used multiple operationalizations of customer perceived quality to discover the trends that are common across them (we also used multiple measures because it is impossible to reflect various failure characteristics in a single number).

We performed internal validation of the data by obtaining and comparing metrics for several data sources. For example, we considered both calendar time since GA and total system runtime. Our comparisons showed that both predictors had similar effects.

To validate our data extraction process, we independently wrote programs to extract and process data and conducted independent analysis to make sure that the results are not due to mistakes in data processing and analysis. Data analysis was conducted as a pipeline where raw data was imported from operational databases in the first stage, the relevant filtering was performed in the second stage, summaries were produced in the third stage, and statistical analysis in the last stage. The analysis code included several thousand lines of Perl and R code and small amount of SQL and shell script code.

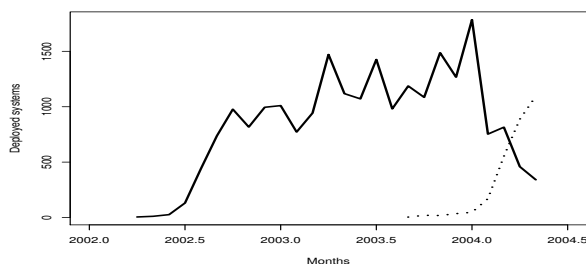
Validation and test data sets were constructed to verify the accuracy of each stage.

To make sure that data is homogeneous we used nuisance factors to separate data sets that were reported or collected using a different process or by a different organization.

Spearman correlations between predictor factors were mostly low. The top correlation of  $-.89$  was between  $nPortNA$  and  $\log(nPort)$ . The only other correlation above  $.5$  was between  $Svc$  and  $nPort$   $0.51$ . These correlations with nuisance factors indicate that the probability that the number of ports is recorded in equipment databases is more likely if

there is a service contract, and, obviously, that the number of ports is either available or not, leading to a large negative correlation. Inspection of regression residuals also did not reveal unusual patterns.

Each organization or software system may be deployed in a different manner that would affect the type of models that may be most suitable. For example, shrinkwrap software may be introduced quite broadly to thousands or millions of users, while the software investigated here is introduced more gradually. Figure 3 shows number of newly deployed systems over time for two major releases we investigate. The counts include brand new systems and upgrades to a new release of existing systems.



**Figure 3. Deployment of systems each month**

Despite the attempts to validate the results it is important to note that they represent one, albeit, very large project and organization and may not generalize elsewhere. We tried to make the analysis as transparent and applicable to other projects and organizations and expect to learn more from applying it elsewhere. We further encourage such application by trying to address questions of significant business importance including predicting customer experiences and predicting staffing needs. Indeed, initial application of the approach in two other Avaya projects is bringing excellent results.

Of course, the ultimate validation is the quality and utility of the prediction in practice.

## 6. Measurement using project support systems

The utility of the presented models in other projects can be shown if the model is used in project planning or project understanding. In this section we outline the steps needed to apply the model in a software project. There are four basic stages: project data extraction, project data validation, and modeling of relevant phenomena.

In the data extraction stage access to the project systems is obtained and raw project data is extracted. In case of home-grown tools, it may be necessary to interview a person responsible for tool support to understand the structure and functionality of such systems.

IBM has data from a tiered customer support system and change management systems similar to the one at Avaya.

Buckley and Chillarege have described the RETAIN database as well as license tracking systems at IBM that capture call center information, developer defect information, and software license information [4]. In fact, more detailed information is available for modeling at IBM in process and databases that uses the Orthogonal Defect Classification method [5, 12].

HP's NonStop Enterprise Division (NED) has a customer support system and software polling system that produces data similar to Avaya's [18]. HP's support organization has information about non-defect related support requests for HP NED products as well as software failures. NEDs uses installation data and software polling to determine the installed base. Product related quality factors can be provided by the development organization.

The analysis based on data produced by project support tools and databases has a number of distinct benefits that may not be immediately obvious. The data collection is noninvasive, using only existing data and making analysis possible in commercial projects that are usually under intense schedule pressure and do not have time or resources to collect additional data. Long history on past projects is available, enabling comparison to what happened in the past and customization and calibration of the methods to the existing environment. The information is often fine grained, at the problem report/customer call/software change level. The information is complete, all issues and artifacts that are supported by project tools are recorded. The way the project support systems, tools, and databases are used rarely changes, making data uniform over time. Even small projects generate large volumes of data making it possible to detect even small effects statistically. The project support systems are used as a standard part of the project, so the development project is unaffected by experimenter intrusion eliminating observer effects.

## 7. Related work

Our work differs from previous works in three areas. First, we quantify and predict quality perceived by a single customer. Second, our models predict quality measures for a widely-deployed commercial system and can be used to predict current staffing needs. Finally, we look at a broad range of customer perceived quality measures and quality factors.

Much of the prior research in software reliability has been conducted on systems where the testing environment and the deployment environment are similar. Similarities like hardware platforms, software configurations, and usage patterns have allowed researchers to extend defect models from development into the field. Lyu in [12] provides a comprehensive review of previous works. Lyu explains model origins, states modeling assumptions, and classifies commonly used reliability models. We consider a commercial software system that is widely deployed and is used by many customers.

We have neither prior knowledge nor control over environmental factors. These differences in the systems make much of the methodology of prior researches inapplicable. In addition, we take a broader view of customer quality and consider many other quality measures in addition to defect occurrences.

Our work is similar to efforts to certify software. Voas advocates certifying commercial software for use in a customer's environment [16], and Wallnau et. al. at the Software Engineering Institute are conducting research on predictable assembly from certified components [11]. Both approaches test software in the customer's environment then extend results into usage. These approaches account for several environmental variables and make statistical guarantees about various properties. However, we feel that the cost is prohibitively expensive. In order to help guide staffing needs for the development organization, every customer's setting needs to be tested, which is equivalent to exhaustive testing. With thousands of customers, schedule constraints, and resource constraints, exhaustive testing is unfeasible. Our approach does not require intrusive individual customer testing and we examine more quality measures.

Our work is related to previous works that examine quality measures linked to customer perceived quality. Buckley and Chillarege at IBM [4] examined customer surveys to determine the important indicators of customer satisfaction. Basan and Santhanam at IBM have also examined factors that are related to customer perceived quality [2]. However, neither works have attempted to predict the quality measures.

Related works that attempt to measure the number of faults in a system and aid the software development organization allocate resources have not focused on single customers and have not included a wide range of quality measures.

The COQUALMO project at USC uses COCOMO II data to estimate the total number of defects in a software system [6]. Their model uses size metrics and various process modifiers. The process modifiers measure aspects of the defect injection process and the defect removal process. They have not applied their method on an active software development project. Jones et al. [9] uses software product metrics and usage data to predict the likelihood of a defect occurrence for a large telecommunications system at Nortel. The authors collected deployment records and computed the usage for a module as the percent of deployed systems with the module installed. The authors also computed various software content factors including call graph metrics, control flow metrics, and statement metrics. A logistic regression model was used to evaluate the probability of a defect occurrence and the significance of the predictors. Results show usage to be a statistically important predictor, but that distinct include files and log base 2 of the number of independent paths were more statistically significant. The authors point out that including development process information can produced better results. Li et. al have compared defect-occurrence data from multiple

releases of four widely-deployed production software systems, including commercial and open source systems [10]. They find that the Weibull model is the preferred model for modeling defect occurrences for the systems in their study, naïve parameter extrapolation methods are inadequate, and hypothesizes that various factor including ones mentioned in this paper drive the defect-occurrence pattern.

## 8. Discussion

We present models that can predict customer experiences for a single customer and can predict aggregated customer interactions. The results are encouraging, showing that prediction is not only possible, but also accurate enough to guide important business decisions related to targeted deployment that could improve customer perceived quality. The models also quantify relative importance of delivery schedule, complexity of usage, and use of a platform in shaping customer perceived quality. These results can help guide quality improvement efforts.

Our wide variety of measures also allows for customer support planning by customer support organizations. In addition to traditional software defect occurrence predictions which helps resource planning at the development organization level, we also provide customer call and technician dispatch predictions which aids in the planning for direct customer support.

Finally, we present methodology and caveats on how to use software and customer support repositories to facilitate replication of our results in other environments.

Our models show that some measures of quality perceived by a customer can vary by up to 30 times for the highest availability systems depending just on the manner of deployment. This indicates profound importance of deployment strategy in managing customer perceived quality, especially in cases when customer expectations are highest. It is important to note that the complexity of configuration and environment for the products we analyzed makes it impossible to replicate all customer environments during system verification. Surprisingly, the predictors affect all five customer perceived quality measures in similar ways, i.e., the change of the predictors that is associated with improvement of one measure is also associated with improvements in the other measures. The knowledge of such large differences and simple relationships between predictors and the five customer perceived quality measures allows making business decisions that have significant impact on the project. We plan to report applications of such models to support release planning and improve software quality in this and in other projects.

## References

[1] D. Atkins, T. Ball, T. Graves, and A. Mockus. Using version control data to evaluate the impact of software tools: A case study of the version editor. *IEEE Transactions on Software Engineering*, 28(7):625–637, July 2002.

[2] K. Bassin and P. Santhanam. Use of software triggers to evaluate software process effectiveness and capture customer usage profiles. *Eighth International Symposium on Software Reliability Engineering*, pages 103 – 114, 1997.

[3] N. Breslow. Statistics in epidemiology: the case control study. *JASA*, 91(433):14–28, 1996.

[4] M. Buckley and R. Chillarege. Discovering relationships between service and customer satisfaction. *Proceedings of the International Conference on Software Maintenance*, pages 192 – 201, 1995.

[5] R. Chillarege, S. Biyani, and J. Rosenthal. Measurement of failure rate in widely distributed software. *Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 424–433, 1992.

[6] S. Chulani. Coqualmo (constructive quality model) a software defect density prediction model. *Project Control for Software Quality*, 1999.

[7] S. R. Dalal and C. L. Mallows. When should one stop testing software? *Journal of American Statist. Assoc.*, 83:872–879, 1988.

[8] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(2), 2000.

[9] W. Jones, J. Hudepohl, T. M. Khoshgoftaar, and E. B. Allen. Application of a usage profile in software quality models. *Third European Conference on Software Maintenance and Reengineering*, pages 148–157, March 1999.

[10] P. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam. Empirical evaluation of defect projection models for widely-deployed production software systems. *SIGSOFT 2004 / FSE-12*, November 2004.

[11] P. L. Li, M. Shaw, K. Stolarick, and K. Wallnau. The potential for synergy between certification and insurance. *International Workshop on Reuse Economics in conjunction with ICSR7*, April 2002.

[12] M. R. Lyu. *Handbook of Software Reliability Engineering*. IEEE Society Press, Los Alamitos, CA, 1996.

[13] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[14] A. Mockus, D. M. Weiss, and P. Zhang. Understanding and predicting effort in software projects. In *2003 International Conference on Software Engineering*, pages 274–284, Portland, Oregon, May 3-10 2003. ACM Press.

[15] J. Musa, A. Iannino, and K. Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGrawHill, New York, 1987.

[16] J. Voas. User participation-based software certification. *Proceedings of Eurovav 1999*, pages 267–276, June 1999.

[17] S. Weisberg. *Applied Linear Regression, 2nd Edition*. John Wiley & Sons, USA, 1985.

[18] A. Wood. Software reliability from the customer view. *IEEE Computer*, pages 37–42, August 2003.

[19] P. Zhang, J. Landwehr, and M. Serban. Quantifying the value of remote maintenance: An analysis of customer outage data, 2004.