Predictors of 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 direct quality improvement efforts to maximize the 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 platform, and hardware configurations. We predict customer perceived quality as measured by various service interactions, including software defect reports, requests for assistance, and field technician dispatches using the afore mentioned and other factors for a large telecommunications 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 effects of deployment schedule, hardware configurations, and software platform can increase the probability of observing a software failure by more than 20 times. Furthermore, we find that the factors affect all quality measures in a similar fashion. Our approach can be applied at other organizations, and we suggest methods to independently validate and replicate our results.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Quality, Metrics, Modeling

1. INTRODUCTION

Anticipating customers' experience with a new software release is of significant business importance. The decision to release software early may be necessary because of competitive pressures or the need for additional revenue. However, 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 the first three months of installation to facilitate maintenance 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 importance of various process and product factors on customer experience, which can help guide quality improvement efforts to maximize the return on investment. The factors we examine are: deployment issues, usage patterns, software platform, and hardware configurations.

Finally, we want a method that can be easily adapted and used at other organizations. Therefore, we attempt 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 our results can be independently validated.

Our approach is first to design and operationalize a small set of customer experience measures and product/process factors, then to create and validate customer experience models based on these measures and factors, and finally to produce predictions that answer questions essential to a software company. Our primary motivation is 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 customer reported issues?

We use measures and factors gathered during the customer support process from systems that track customer installations, updates, and complaints. We also use measures and factors gathered during development from systems that track and manage software development activities.

To operationalize customer perceived quality measures we spend a significant amount of effort familiarizing 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 apply various validation techniques to ensure the accuracy of the extracted measures and factors.

We find predictors measuring system installation date, operating system, upgrades, and system size to be significant in determining the customer experience and predicting resources needed to handle customer issues. We fit a logistic model and four linear regression models to quantify the relative importance of eight predictors on four measures of customer perceived quality and to enable informed decisions regarding deployment strategies and staffing

ICSE'05, May 15–21, 2005, St. Louis, Missouri, USA.

needs. The predictive value of the models varies based on the frequency of the predicted event: rare events like a customer reported problem that causes a change in the software are more difficult to predict than frequent aggregated events like customer calls in the first three months of installation.

We start by describing our motivation in Section 2. Background on the project under study and the analysis method are in Section 3. Section 4 introduces our models of customer experience and Section 5 presents our findings. Section 6 discusses the validity of our results and Section 7 outlines how similar prediction techniques can be applied at other organizations. We conclude with related work in Section 8 and discussion in Section 9.

2. MOTIVATION

While there are techniques to predict how many faults remain in an unchanging software system (see, e.g., [14, 7, 11]), which modules (see, e.g., [8, 9]) or changes (see, e.g., [12]) will have defects, and how much effort defect repairs will require (see, e.g., [1]), the release of a software system is typically full of unknowns: when is the quality finally good enough, what will a particular customer experience, how much resources will the maintenance support require? In this paper, we focus on customer experience prediction models that quantify the effects of various factors on customer experiences and to provide a quantitative basis for making business decisions.

Our findings can enable a company to assess the quality being delivered to a specific customer and can allow for deployment or other adjustments to ensure that the quality expectations of the customer are met.

Knowing the influence of each factor on customer perceived quality can also allow for targeted improvements in the development process, the support processes, or the software itself, to maximize the return on improvement efforts.

We also address resource planning issues. We predict customer calls, technician dispatches, and field software defects, which can be aggregated to predict customer support and maintenance resource needs.

Many research works have examined the effects of software content and development process on measures of customer perceived quality. However, few works have considered hardware configurations, software platform, usage patterns, and deployment issues. The end users of a software typically experience the quality of the entire “solution”, which includes physical systems, terminals, and networks, as well as the “pure” software platform which includes 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 failures in the software under study. Human factor problems such as software upgrade and configuration often yield their own significant share of failures.

Prior work have assumed that hardware configurations, software platform, usage patterns, and deployment issues are important, but few research work have attempted to validate the claim or to quantify the effects of the factors. Musa emphasizes the importance of establishing an operational profile that considers hardware configurations, software platform, usage patterns and deployment issues in [14], but provides no evidence of the factors’ importance.

Determining the quality delivered by a software product is a complicated task, given research indicating that customer satisfaction is far more complex than merely minimizing the number of defects in a system [3, 6]. Our research seeks to capture and predict multiple aspects of customer perceived quality in quantitative models that can explain and order various factors in terms of their effect on customer experience.

To determine the various measures and factors, we rely on data that established companies with a large customer base already capture during customer support and product development processes. We use quality measures that are similar to those used by other large established companies such as IBM and HP [3, 6, 17]. The factors we consider have been cited as important in previous work such as [9] by Jones et. al. and [11] by Lyu .

3. BACKGROUND

In this section, we describe the context of our study. We describe the software project, the systems that track development, and the systems that track installed products. We also briefly describe the methodology used to extract data from the databases.

3.1 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 software development organization deploys major releases on a fixed schedule, with subsequent dot releases that bundle patches and refinements to the system.

Multiple 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.

We use customer interaction measures and factors captured in four databases. Two databases contain customer information including service request information (e.g., trouble tickets) captured during the post-sale customer support process and customers’ product information. The other two databases contain information captured during product development including change requests and code changes, which we use to identify customer reported problems that resulted in software changes.

The project under consideration uses the Sablime system for problem tracking and an internal version control system. Modifications as a result of customer interactions described below can be traced back to the customer and the related interaction. Details on the nature and analysis of software change data is in [13].

3.2 The customer support process

Avaya uses a tiered support process similar to those in other organizations [17, 3], and has smart agents deployed on most products. Our data come from the trouble ticket database and the equipment database.

The trouble ticket database contains information about customer contacts. A trouble ticket is generated for each customer contact, whether it is an alarm sent by the smart agent or a phone call from a real person. Each ticket contains 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 contract the customer has. Roughly half of the over 4 million tickets created in 2003 are related to products we analyze in this paper.

The equipment database contains information about products installed at each customer location, regardless of whether the product has reported any problems. 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 systems listed in the equipment database, but only around 100K systems contain products we consider in this paper.

4. MODELS OF CUSTOMER PERCEIVED QUALITY

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. This is consistent with prior findings at IBM [3, 6]. 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. In this paper, 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 infrequent problems that are costly in terms of time and resources needed to resolve and the interruption to the customers' business.

A customer reported failure that leads to a software change is an example of this type of problem. It is costly to the customer because of the time needed to escalate the problem through the tiered customer support organization, diagnose and fix the problem, and delivery the fix via a patch to the customer. The problem remains unresolved for a long period of time which may reduce the functionality of the system for the customer. Chulani et. al. in [6] shows that this type of measure is related to customer satisfaction.

We are also interested in frequent low impact problems, which are problems that may have a low per incident impact, but which, if occur frequently, may negatively impact customer perceived quality. A customer call into the support center is an example of this class of problems. A single call may not be a major interruption to the customer. However, a large volume of calls may indicate a serious issue. Buckley and Chillarege [3] shows that this type of measure is related to customer satisfaction.

In the following sections we define and operationalize measures and predictors of customer perceived quality based on data captured in the project monitoring and customer support systems. Similar data are available for most high availability business-critical software systems and are discussed in Section 8.

4.1 Measures of customer perceived quality

We consider measures that are extracted from Avaya's customer support systems, which have been developed and improved over several decades. The measures are:

- 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 a slightly different aspect of the customer experience. The measures reflect experiences that vary in severity and amount of time until resolution.

We have chosen to model customer interactions in the first three months of system installation, because this is a reasonable amount of time to set up, configure, and tune even a very sophisticated software system such as a switching software. The initial period is most fraught with risks, therefore of the most concern to the software provider and to the customer. For example, the probability of a custumer reporting a software issue was several times greater in the first three months after installation than in the subsequent three months.

4.2 Predictors of customer perceived quality

We examine deployment issues, usage patterns, software platform, and hardware configurations. We use several measures, which we call *predictors*, to measure these factors. Ideally, we would like one-to-one or many-to-one mappings between the predictors and a factor so we can isolate the effects of each factor. However, such perfect measures are rarely available in empirical studies. The predictors sometimes measure several factors.

For example, the number of ports on an installed system, measures the usage patterns factor and the hardware configurations factor (since some hardware components are usually associated with larger systems). However, there are substantial variations in the number of ports even within systems with similar hardware configurations, so the number of ports provides information about the system not captured by other predictors.

4.2.1 System size

The system size predictor measures the hardware configurations factor, software platform factor, as well as the usage patterns factor. We consider systems running on small to medium and large platforms. 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 whether the systems is on a small to medium or a large platform using an indicator variable called LARGE.

We expect small to medium systems to have fewer customer interactions. First, there are fewer settings to configure and fewer systems to interface with. Second, smaller systems may have less usage than larger systems. Finally, smaller systems may not be as likely to be involved in business critical applications that require 7x24 uptime. Consequently, customers of smaller systems are less likely to experience and report issues.

4.2.2 Operating system

The operating system predictor measures the software platform factor and the hardware configurations factor. We consider systems running on a proprietary, an open (Linux), and a commercial (Windows) operating system. Small, medium, and large systems use the proprietary and the open operating systems. Only a very small version of the system uses the WindowsNT/Windows2000 operating system. In our models, we encode the three operating systems using two indicator variables, OX and WIN.

We expect off-the-shelf operating systems (Windows and Linux) to introduce unnecessary complexity and configuration issues that can be more easily controlled in a proprietary system (encoded as OX), where only essential features are supported and versions and configurations can be precisely tested and tuned to reflect deployed environments.

4.2.3 Ports

The ports predictor measures the usage pattern factor and the hardware configurations factor. The number of ports indicates how many licensed endpoints are supported by the system. The usage patterns are likely to be very different for machines with different number of ports and some equipment is only available for systems with more ports. In our models, we encode the log number of ports with the log(nPort) variable.

We expect systems with more ports to be have more customer interactions. A system supporting only 100 users is less likely to experience a problem compared with a system supporting 3000 customers that is likely to be operating near its operational limits. This is due to both the increased amount of usage and the increased amount of usage at borderline and complex situations.

4.2.4 Total deployment time

The deployment time predictor measures the deployment issues factor. We use total system runtime on all deployed systems from the installation of the first system until the installation of the j^{th} system as a measure of deployment time:

$$Runtime(t_j) = \sum_{t_i < t_j} (t_j - t_i)$$

where t_i is the installation times of the i-th systems. In our models, we encode the log of the total deployment time using the $\log(rtime)$ variable.

As a new release is used by customers and exposed to the varied usage patterns, more issues may be reported. These issues are then fixed via patches and incorporated in the later dot releases. Since these dot releases are shipped later in a major release's deployment cycle, systems installed later will not experience problems detected by early customers. These systems may demonstrate better quality. As Avaya gathers experience in detecting and remediating problems and as technician and customer support teams improve product knowledge through installation and configuration, field personnel will become better in helping customers avoid problems. Therefore, we expect fewer customer interactions as the total deployment time increases.

4.2.5 Software upgrades

The software upgrades 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 our model we encode the existence of an upgrade using an indicator variable called Upgr.

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.

4.3 Nuisance factors

In addition to the predictors in Section 4.2, it is clear that customer reporting practices and organizational factors also affect customer interactions. These 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 our analysis, we include variables that we think will have significant effects on the measures of customer perceived quality. We call them nuisance factors because they are likely to identify peculiarities of data reporting and collection process, but not necessarily differences in the underlying customer perceived quality.

4.3.1 US or international installation

We make a distinction between US domestic and international customers because the support processes differ significantly. There may be other differences, for example, the system may interfaces with slightly different equipment and networks. In our models, we encode the location of the installation using the indicator variable called US.

4.3.2 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 the customer requests help, the customer may be charged a fee depending on the nature of the problem, plus the appropriate parts and labor charges. We speculate that customers without a service contract agreement tend to report issues less frequently. By contrast, customers with a service contract tend to report problems more often. In our models, we encode the existence of a service contract using the indicator variable called Svc.

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

This measure may be confounded with customer types. Customers that require very high availability are more likely to pay for a full coverage service agreement.

4.3.3 Missing configuration information

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

5. RESULTS

In this section, we present results of our regression analysis. The response variables are in the form of occurrence counts except for the very rare event of a 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 [16].

We first fit models to test the relationships hypothesized in Section 4 using the data from a single major release. Then we use the models 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 4 and briefly discuss the results for the other measures.

5.1 Software failures

We attempt to predict if a customer will observe a failure that leads to a software change using logistic regression. Our response variable Y_i^{MR} is binary. One, if a customer observes a failure that leads to a software modification within the first three months of system installation and zero otherwise. Our predictor variables, $\tilde{x}_i$ are described in Section 4.The model is:

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

5.1.1 *Modeling software failures*

	Estimate	Std. Err.	z-value	Pr(>|z|)
(Intercept)	-5.26	0.64	-8.18	$3 * 10^{-16}$
$\log(rtime)$	-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 4.2, decreases the probability that a customer will observe a failure that leads to a software change. Existence of upgrades (Upgr) increases the probability. A customer using the Windows platform has the highest probability. The next most likely is the Linux platform. The least likely is the proprietary platform. The probability increases with the number of ports (nPort) and also where the number of ports is not reported (nPortNA). Large systems (LARGE) have a higher probability. Finally, the two nuisance factors indicate that customers with service contracts (Svc) and in the United States (US) are more likely to observe a failure that leads to a software change.

The model reduces the deviance by about 400, the residual deviance is still quite high: around 2000, indicating that there is still much unexplained variation. This is not particularly surprising since development, verification, deployment, and service processes are all designed to eliminate failures. Consequently, if there are obvious causes of failures, then the relevant organizations will have taken measures to address the issues. The result is a more random failure occurrence pattern.

The model indicates that the total deployment time is one of the most important predictors of observing a failure that leads to a software change. It is important to understand why such a relationship exists. Customers who installed the application early may have detected malfunctions that are fixed by the time later customers install their systems. In addition, the individuals performing the installation and configuration may have acquired more experience, have access to improved documentation (by documentation we also have in mind emails, informal conversations, and discussion lists), and have better training, which increase the awareness of potential problems and work-arounds.

The lesson from this relationship is that customers that are less tolerant of availability issues should not be the first to install a major software release. This is a well known practice that is often expressed as a qualitative statement: “never upgrade to dot zero release.” The probability of observing a failure that leads to a software change drops from 13 to 25 times for the most reliable proprietary operating system as runtime goes from zero (the first system installed) to the time that is at the midpoint in terms of the rtime predictor, depending on system configuration. The least reliable Windows platform experiences a drop in probability of 4 to 8 times and the Linux platform experiences a drops of 7 to 24 times depending on configuration. This indicates that for the most reliable software platform the deployment schedule has a tremendous impact on the probability that a customer will experience a failure that leads to a software change.

The number of ports is significant even after adjusting for the system size. Since the number of licensed ports also represents system utilization, we may infer that the amount of usage is important in predicting the probability of observing a failure that leads to a software change as hypothesized.

The only surprise is that the existence of an upgrade is related to a higher probability. This suggests that upgrades may be the manifestation of system complexity, which increases the probability of both upgrades and failures.

Nuisance parameters require slightly different interpretations because they distinguish among populations of customers and different reporting processes. The positive coefficient in Table 1 for the Svc variable may appear to be counterintuitive because having a service agreement should help reduce the probability (a negative coefficient for Svc). Our interpretation is that having a service agreement significantly increases a customer’s willingness to report minor problems, which biases results (for more detail see [18]). 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., [2], in the statistical literature.

5.1.2 *Predicting software failures*

To demonstrate the applicability of our model we predict software failures that lead to a software change for a new release using the model fitted from previous releases (in this case one previous release). We refit the model in Table 1 for prediction using the most significant predictors with p-values below 0.01 as shown in Table 2. We also exclude the predictors log(nPorts) and nPortNA because the predictors are not available to us at the time of analysis for the new release.

	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 from Table 2 estimated from data on the old release and predictors for the customers deploying the new release to predict the probability of a customer experiencing a failure that leads to a software change. Customers with predicted probability of failure above a certain cutoff value c are predicted to experience a software related failure. These predictions are then compared with actual reports. The predictions are characterized using Type I (the proportion of systems that do not observe a failure but that are predicted to observe a failure) and Type II (the proportion of systems that observe a failure but that are not predicted to observe a failure) 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 quantiles of the probability (fraction of customers that 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 conduct a cost-benefit analysis of the cut-off values. The decision may be different

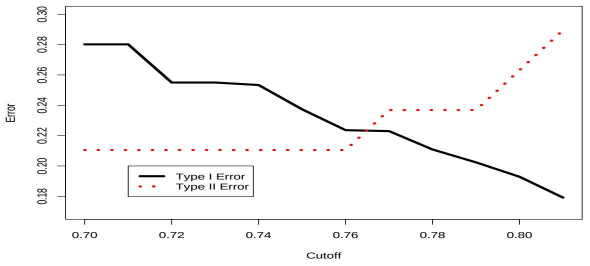

Figure 1: Type I and II errors.

for different products and customers. A higher cut-off may satisfy customers that are less tolerant of failures. A lower cut-off may satisfy customers that are aggressively exploring new capabilities.

5.2 Customer calls and other quality measures

We attempt to predict the number of calls, system outages, technician dispatches, and alarms within the first three months of installation using linear regression. For example, in the case of calls, the response variable Y^{calls} is the number of calls within the first three months of installation transformed using the log function to make errors more normally distributed. The predictor variables, $\tilde{x}_i$ are described in detail in section 4. The model is:

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

5.2.1 *Modeling customer calls*

	Estimate	Std. Err.	t value	Pr(>|t|)
(Intercept)	0.35	0.04	7.90	$3 * 10^{-15}$
$\log(rtime)$	-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 are statistically significance due to large sample sizes. Table 3 shows the fitted coefficients for the number of calls. The regression results for the other three quality measures (system outages $R^2 = .06$, technician dispatches $R^2 = .15$, and alarms $R^2 = .18$) are similar except for the cases discussed below.

The total deployment time factor improves (decreases the number) all five quality measures and is highly significant. The existence of an upgrade (Upgr) makes all five quality measures worse. As we discussed in Section 5.1.1, upgrades may be confounded with complexity. Larger (LARGE) systems are worse than small to medium systems across all measures. Increase in the number of ports (nPorts) degrades quality with respect to all five measures.

The operating system did not always have the hypothesized effect. The numbers of outages, dispatches, and calls are lower for Linux than for the embedded system. We do not have a good explanation of this discrepancy. Furthermore, the Windows platform has worse quality measures except for the number of alarms. However, this is not surprising since only a few types of alarms are generated on low end systems running on the Windows platform.

Despite the few exceptions, it is reassuring to see the diverse measures of customer perceived quality being affected in almost the same fashion by the factors. This implies that, at least in terms of these measures, different aspects of quality do not need to be traded-off against each other. Simultaneous improvements in all measures of customer perceived quality are possible.

5.2.2 *Predicting customer call traffic*

We demonstrate the applicability of our model by predicting customer call traffic from new customers for a new release to help determine staffing needs of a customer support organization. On average, each customer support specialist can process a fixed number of calls per month. Therefore, to predict staffing needs, it is sufficient to predict the total number of calls per month. We refit the model in Table 3 without the predictors log(nPorts) and nPortNA to predict the number of calls from new customers for a new release. Figure 2 illustrates the trend of predicted and actual inflows of calls.

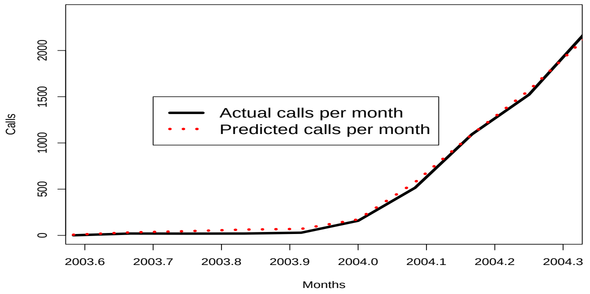

Figure 2: Prediction of monthly call traffic.

The two trends are very close to each other indicating that the flow of calls can be predicted fairly accurately. Due to space limitations we do not present full details of predicting the inflow of calls for new and existing systems.

6. VALIDATION

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

We 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 and field personnel to ensure that results are consistent with their perception of reality. We used multiple operationalizations of customer perceived quality to discover common trends (we also used multiple measures because it is impossible to capture various aspects of customer perceived quality using a single measure).

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

To validate our data extraction process and analysis, we independently wrote programs to extract and process data and performed independent analysis. Data analysis was conducted as a pipeline process 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 was performed in the last stage. The analysis code included several thousand lines of Perl and R code and small amounts of SQL and shell script code.

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

To ensure that data was homogeneous we used nuisance factors to separate data sets that were reported or collected using different processes or by different organizations.

Spearman correlations between predictors were mostly low. The top correlation of $-.89$ was between nPortNA and $\log(nPort)$. This relationship was obvious. The port information was either available $\log(nPort)$ or not nPortNA. The only other correlation above .5 was between Svc and nPort 0.51. The correlation indicated that the number of ports was more likely to be recorded in equipment databases if the customer had a service contract. Inspection of regression residuals did not reveal any unusual patterns.

Each organization or software system might be deployed in a different manner that would affect the type of models that may be most suitable. For example, shrinkwrap software might be introduced quite broadly to thousands or millions of users, while the software investigated here was introduced more gradually.

Figure 3 shows number of newly installed systems over time for the two major releases we investigate. The counts include new systems and upgrades. Despite attempts to validate the results, it is important to note that they represent one, albeit, very large project and may not generalize elsewhere. We try to make the analysis transparent and applicable to other projects and organizations. We expect to learn much from applying the method elsewhere. We encourage application of the results to address questions of significant business importance including predicting customer experiences and predicting staffing needs. The ultimate validation is the quality and utility of the prediction in practice. Indeed, initial application of the approach in two other Avaya projects is bringing excellent results.

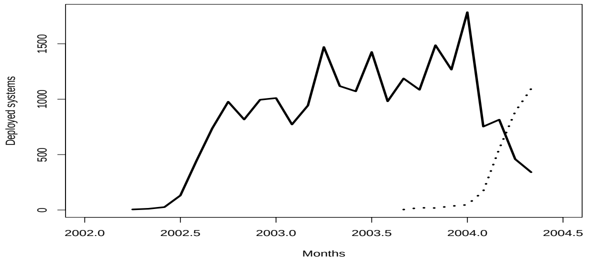

Figure 3: Deployment of systems each month

7. MEASUREMENT USING PROJECT SUPPORT SYSTEMS

In this section we outline the steps necessary to replicate results at another organization. There are three basic steps: project data extraction, project data validation, and modeling of relevant phenomena.

In the data extraction stage access to the project support and project management 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.

We identify data sources that are available at other organizations that can be used to replicate our results. IBM has data from a tiered customer support tracking system and change management systems similar to the one at Avaya. Buckley and Chillarege describe the RETAIN database as well as license tracking systems at IBM that capture call center information, defect information, and software license information [3]. In fact, more detailed Orthogonal Defect Classification information is available at IBM [4, 11].

HP's NonStop Enterprise Division (NED) has a customer support tracking system and a software polling system that produces data similar to Avaya's [17]. HP's support organization has information about non-defect related support requests for HP NED products as well as software failures. NED uses installation data and software polling information to determine the installed base.

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 already exist and data collection is non-intrusive, which makes analysis possible in commercial projects that are usually under intense schedule pressure and do not have time or resources to collect additional data. History on past projects is available, which enables comparison to what happened in the past and enables customization and calibration of methods to the 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 project support systems, tools, and databases are used rarely changes, which makes data uniform over time. Even small projects generate large volumes of data, which makes it possible to detect even small effects statistically. Project support and tracking systems are used as a standard part of the project. The development project is unaffected by experimenter intrusion, which eliminates observer effects.

8. RELATED WORK

Our work differs from previous work in three ways. 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 examine a broad range of customer perceived quality measures and factors.

Related work like [8, 12, 9, 5] predict the total number of faults in a system or if a component will be defect prone. The results aid maintenance resource planning and guide testing. However, previous work have not focused on a single customer, have not included a wide range of quality measures, and have not considered deployment issues, software platform, or hardware configurations.

Many works have considered content factors and development factors, likes lines of code, Cyclomatic complexity, code age, and the number changes to implement a feature, to predict defect occurrences. One such effort is the COQUALMO project at USC, which is the quality extension of the COCOMO II project [5]. The COQUALMO model uses size metrics and various process (development process) modifiers, to predict the total number of residual defects in a software system using a linear model. However, COQUALMO does not consider deployment issues, hardware configurations, software platform, or usage patterns and predicts only the total number of defect. Jones et. al. [9] uses usage patterns data in addition to software product metrics to predict the probability of a defect occurrence for a large telecommunications system at Nortel. The authors use a logistic regression model to predict the probability of a defect occurrence and to quantify the impact of the predictors. Results show usage to be a statistically important predictor. However, Jones et. al. do not consider deployment issues, hardware configurations and software platform. The predictions indicate the probability that a module contains a defect and not the probability that a customer will experience a defect. Neither work predict other important customer interactions.

Our work is related to previous work that examine quality measures linked to customer perceived quality. Buckley and Chillarege at IBM [3] examine customer surveys to determine that the number of defect fixes and the total number of problem reports have the most influence on customer satisfaction for one IBM product. Chulani et. al. in [6] examine customer satisfaction surveys across multiple IBM products to find that the amount of time to resolve an issue is more closely related to customer satisfaction than the number of code related defects. We use their results to determine our measures of customer perceived quality.

Our focus on quality as perceived by a single customer is similar to efforts to certify software. Voas advocates certifying commercial software for use in a customer's environment [15], and Wallnau et. al. at the Software Engineering Institute are conducting research on predictable assembly from certified components [10]. 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 their cost is prohibitively expensive. In order to help guide staffing needs for the development organization, every customer's setting needs to be tested. With thousands of customers, schedule constraints, and resource constraints, exhaustive testing is infeasible. Our approach does not require intrusive individual customer testing and we examine wide range of customer interactions.

Much of the prior research in software reliability has been conducted on systems where the testing environment and the deployment environment are similar. Similarities in hardware configurations, software platform, and usage patterns have allowed researchers to extend defect occurrence patterns from development into the field. Lyu in [11] provides a comprehensive review of previous works, including model origins, modeling assumptions, and a classification of commonly used reliability models.

Our approach is different from prior approaches that extend results from testing into the field. 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. This makes extending results from testing to the field infeasible. In addition, we take a broader view of customer perceived quality and consider many other quality measures in addition to defect occurrences.

9. DISCUSSION

We present models that can predict customer experience for a single customer and can predict aggregated customer interactions. The results are encouraging, showing that predictions are not only possible, but are also accurate enough to guide important business decisions including targeted deployment that could improve customer perceived quality. The models also quantify the relative importance of delivery issues, usage patterns, software platform, and hardware configurations in shaping customer perceived quality. These results can help guide quality improvement efforts.

We predict a wide variety of measures that allows for customer support resource planning by customer support organizations and for maintenance resource planning by the software development organization. In addition to traditional software defect occurrence predictions, which help resource planning at the development organization level, we also predict customer calls and technician dispatches, which aid planning for direct customer support.

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

Our models show that some measures of customer perceived quality can vary by up to 30 times for the highest availability systems just depending on the manner of deployment. This indicates the profound importance of deployment strategy in managing customer perceived quality, especially when a customer's expectations are high. It is also important to note that the complexity of configurations and environments 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., a change in the predictors associated with improvement in one of the quality measures is also associated with improvements in the other quality measures. The knowledge of such large differences and simple relationships between predictors and the five customer perceived quality measures allows making decisions that can have significant impact on the project. We plan to report applications of our models to support release planning and to improve software quality in this and in other projects.

10. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grants ITR-0086003 and CCF-0438929, by the Carnegie Mellon Sloan Software Center, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298. We would like to thank all the people in Avaya who provided information directly (via interviews) or indirectly (by working on the products under study.) In particular we thank E. Moritz and D. Sokoler and others for providing insight on customer support process, project management and other aspects of the studied software projects. We would also like to thank Jim Herbsleb and Mary Shaw for their valuable insight and support.

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

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

[4] 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.

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

[6] S. Chulani, P.Santhanam, D. Moore, and G. Davidson. Deriving a software quality view from customer satisfaction and service data. *European Conference on Metrics and Measurement*, 2001.

[7] S. R. Dalal and C. L. Mallows. When should one stop testing software? *JASA*, 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. 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.

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

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

[13] 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.

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

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

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

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

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