Chapter 1

Missing data in software engineering

The goal of this chapter is to increase the awareness of missing data techniques among people performing studies in software engineering. Three primary reasons for this presentation are:

1. the "quick-fix" techniques that drop the cases with missing values may yield biased or inconclusive results. Such techniques are still widely (and often implicitly) used in software engineering;

2. dealing with missing values is no longer a burden for a practitioner, because easy to use statistical software is now available on popular platforms;

3. software represents a distinct data source with unique reasons and patterns for missing data. For example, software studies tend not have the luxury of large sample sizes requiring analysis methods that use all available data, including incomplete cases. Many properties of software can not be measured directly, therefore investigators have to get the necessary information from people who create and maintain a particular piece of software, leading to frequent and complex patterns of missing data.

Section 1.1 discusses sources of software data. The next section introduces an illustrative example evaluating how a software process influences development time. Section 1.3 presents a general statistical perspective for dealing with missing data with an illustrative example. Section 1.4 discusses non traditional missing data problems specific to the field of software engineering. A summary is provided in the last section.

1.1 Sources of software data

Software engineering data come from several distinct sources. The three primary sources are:

- data collected through experimental, observational, and retrospective studies;
- software metrics or reported project management data including effort, size, and project milestone estimates;
- software artifacts including requirements, design, and inspection documents, source code and its change history, fault tracking, and testing databases.

To narrow the scope of the presentation we did not include data sources produced directly by software with little or no human involvement, such as program execution and performance logs or the output of program analysis tools. Such data sources tend to produce tool specific patterns of missing data that are of limited use in other domains.

Surveys in an industrial environment are usually small and expensive to conduct. The primary reasons are the lack of subjects with required knowledge and the minimal availability of expert developers who, it appears, are always working toward a likely-to-be-missed deadline. The small sample size limits the applicability of deletion techniques that reduce the sample size even further. This may lead to an inconclusive analysis, because the sample of complete cases may be too small to detect statistically significant trends. If, on the other hand, the sample sizes are large and only a small percentage of data are missing, a deletion technique (a technique that removes missing observations) may work quite well.

The values in survey data may be missing if a survey respondent declines to fill the survey, ignores a question, or does not know the answer to some of the questions.

Reported data on software metrics often contain the desired measurements on quality and productivity. Unfortunately, the reported data are often not comparable across distinct projects (Herbsleb & Grinter 1998). The reasons include numerous social and organizational factors related to intended use and potential misuse of metrics, and serious difficulties involved in defining, measuring, and interpreting a conceptual measure in different projects.

Reported data need extensive validation to confirm that it reflects the quantities an analyst is interested in. Data collection is rarely a priority in software organizations (Goldenson, Gopal & Mukhopadhyay 1999). The priority of validating

collected data is even lower, often leading to unreliable and misleading software measures. In addition, some software measures are difficult to obtain or have large uncertainty. Examples of such measures include function point estimates or size and effort estimates in the early stages of a project. Frequently data values are missing because some metrics are not collected for the entire period of the study or for a subset of projects.

Software artifacts are large, highly structured, and require substantial effort to interpret. Measures derived from software artifacts tend to be more precise and consistent over time than measures derived from surveys and reported data. They measure the artifact itself, as opposed to the subjective perception of the artifact captured by survey measures. Traditionally, software artifacts are measured based on the properties of source code. Such measures include source code complexity (Halstead 1977, McCabe 1976), complexity of an object oriented design (Chidamber & Kemerer 1994), or functional size (Albrecht & GaffneyJr. 1983). Instead of measuring the source code, it is possible to measure the properties of changes to the code. This requires analysis of change history data, see, for example, (Mockus 2007). Artifact data may be missing or difficult to access for older software artifacts because of obsolete storage or backup media. Consequently, software artifacts are usually available or missing in their entirety, reducing the need for the traditional missing data techniques that assume that data are only partially missing. Measuring such artifacts might require substantial effort, especially if they were maintained using obsolete tools.

1.2 Example Data

To illustrate the application of missing data methods we will use a case study of process improvement in a software organization (Herbsleb, Krishnan, Mockus, Siy & Tucker 2000). The study involved a medium-size, process-oriented software organization performing contract work. One of the study goals was to determine if the excessive detail of software process had increased the development interval. In particular, the study investigated the relationship of development interval and project tracking measures.

The collected data came from three sources: survey questions, reported project metrics, and the source code change history. The development interval was the response or dependent variable. We model (predict) it using several project tracking measures described below that are used as independent, predictor, or explanatory variables.

1.2.1 Survey

A total of 68 surveys of 19 individuals evaluating three dimensions of project tracking process for 42 projects were collected.

The three dimensions of project tracking were defined by the following questions.

1. Were the project's actual results (e.g., schedule, size, and cost) compared with estimates in the software plans?

2. Was corrective action taken when actual results deviated significantly from the project's software plans?

3. Were changes in the project's plans agreed to by all affected groups and individuals?

Subjects evaluated three dimensions of project tracking with ordinal ratings: (1) — "Rarely if ever", 2 — "Occasionally", 3 — "About half of the time", 4 — "Frequently", and 5 — "Almost always". When the subject did not have enough knowledge of the project to answer the question, they entered "don't know".

To exemplify missing data techniques we simplify the analysis by treating each survey as an independent observation. In our example several individuals evaluated most projects and several projects were evaluated by a single individual. Therefore, multiple reports on one project (or done by a single person) are not independent. Unfortunately, adjusting for that dependence would distract from the presentation of missing data techniques.

1.2.2 Software change data

The project interval and size data were obtained from change history databases. The project interval was measured in days from the start of the first change until the completion of the last change. The project size was measured in number of logical changes called Maintenance Requests (MRs).

1.2.3 Reported project data

The reported project data included size, staff months, number of faults, and interval. Unfortunately, reported data were not consistent, therefore it was not used in the models. While some projects measured size in function points (FP), other

projects measured size in lines of code (LOC). The reported function point and LOC measures did not correlate well with the amount of code developed (as obtained from change history) or with the reported staff months of effort. Furthermore, the reported interval did not correlate with the duration of the development phase measured by the time difference between the last and the first change. These serious validity problems made the reported data unsuitable for further analysis.

1.2.4 Missing values

Change history databases for ten of the surveyed projects was moved off line and unavailable for analysis. Because the response variable interval was missing for those projects we excluded them from further consideration (other reasons are given in the discussion of the types of missing data). Additional six cases were dropped because all the project tracking questions were answered "don't know". That left us with 52 cases (corresponding to 34 projects) for the analysis.

The list of data quality problems in this example may seem enormous, but in our experience such data quality is not unusual in a software study.

We used multiple linear regression (see, for example, (Weisberg 1985)) to model the project development interval. The project size and the three tracking measures were independent variables. We included the project size as a predictor because it affects the project interval.

Inspection of the variables showed increasing variances (a scatterplot with a very large density of points at low values) for the interval and size. A square root transformation was sufficient to stabilize the variance of the interval and size and led to the following final model:

$$\sqrt{\text{Interval}} = b_0 + b_1\sqrt{\text{Size}} + b_2\text{Tracking}_1 + b_3\text{Tracking}_2 + b_4\text{Tracking}_3 + \text{Error}. \tag{1.1}$$

The following section describes various techniques to fit such models in the presence of missing data.

1.3 A Statistical Perspective on Missing Data

In statistical analysis the phenomena of interest is commonly represented by a rectangular $(n \times K)$ matrix $Y = (y_{ij})$ where rows represent a sample of n observations, cases, or subjects. The columns represent variables measured for each case. Each variable may be continuous, such as size and interval, or categorical like file or project.

Some cells in such a matrix may be missing. It may happen if a measure is not collected, or is not applicable, for example, if a respondent does not answer a question on a survey form.

The mechanism by which some cells are not observed is important to select an appropriate analysis technique. Denote the response indicator

$$R_{ij} = \begin{cases} 1, & y_{ij} \quad \text{observed,} \\ 0, & y_{ij} \quad \text{missing.} \end{cases} \tag{1.2}$$

Denote all the values of the observations that are missing as Y_{mis} and the rest as Y_{obs}. Let $P(R|Y_{obs}, Y_{mis}, \theta)$ be the probability distribution function of R given a statistical model specified by parameter θ and all the values of Y. The data are *missing at random* (MAR) according to Little & Rubin (1987) if

$$P(R|Y_{obs}, Y_{mis}, \theta) = P(R|Y_{obs}, \theta),$$

i.e., the distribution of the response indicator may depend on the observed values but may not depend on the values that are missing. The data are *missing completely at random* (MCAR) if a stronger condition holds:

$$f(R|Y_{obs}, Y_{mis}, \theta) = f(R|\theta).$$

The MAR assumption allows the probability that a datum is missing to depend on the datum itself indirectly through quantities that are observed. For example, in the described data, the interviewees might remember less about smaller project, resulting in higher likelihood that some of the survey's values are missing. The MAR assumption would apply, because the predictor "project size" explains the likelihood that the value will be missing. MCAR assumption would not apply, because the probability that a value is missing depends on project's size. However, if we do not have a measure of project's size or simply do not include project's size in our estimation model, then even the MAR assumption is not satisfied. Such case is referred to as data not missing at random (NMAR). The NMAR data can be made to satisfy the MAR assumption if variables that characterize situations when a value is missing are added. Therefore, it is important to add variables that might predict the missing value mechanism to the dataset.

Personal income obtained via survey represents a typical example where the MAR assumption is not satisfied. It is well known that extreme values of personal income are less likely to be reported. Consequently, the MAR assumption is violated, unless the survey can reliably measure variables that are strongly related to income. When extreme values are more likely to be missing, the probability

that a value is missing depends on the the value itself and, unless other predictors can fully account for that change in the probability of being missing, the MAR assumption is no longer satisfied.

It is worth pointing out that it is impossible to test the MAR hypothesis based on the dataset itself, since that would require knowing the values for missing observations. It could be tested by gathering additional information, for example, by conducting a repeat survey for the missing cases. However, when the data are missing beyond the control of the investigator one can never be sure whether the MAR assumption holds. It is possible to test the MCAR assumption, (see, e.g. Little 1988, Kim & Curry 1977). However, the MCAR assumption rarely needs to be tested, because the MCAR assumption rarely holds in practice and because many easy-to-use MAR methods are available.

Situations where even the MAR assumption does not hold may require an explicit model for the missing data mechanism. Such methods tend to be problem specific and require substantial statistical and domain expertise. A concept related to NMAR data (even though it is treated separately in literature) involves censoring in longitudinal studies where some outcome may not be known at the time the study has ended. For example, in software reliability we want to know the distribution of time until a software outage occurs. However, at any particular moment in time there may be many software systems that have not experienced an outage. Thus, we only know that the time until the first outage is larger than the current system runtime for these systems, but we do not know its value. A common approach to deal with censored data is to estimate a survival curve using Kaplan-Meier Estimate (Kaplan & Meyer 1958, Fleming & Harrington 1984). The survival curve is a graph showing the percentage of systems surviving (with no outage) versus system runtime. It has been applied to measure software reliability in, for example, (Mockus 2006).

Little & Hyonggin (2003) discuss ways to handle undesirable NMAR data and recommend calculating bounds by using all possible values of missing variables (an approach particularly suitable in case of binary values), conducting a sensitivity analysis by considering several models of how the data are missing, or conducting a Bayesian analysis with a prior distribution for missing values. In most practical situations we recommend attempting to measure variables that capture differences between missing and complete cases in order for the missing-data mechanism to satisfy the MAR assumption. Methods that can handle MAR data can then be applied.

In our example, the “don’t know” answers in survey questions reflect the lack of knowledge by the subject and have no obvious relationship to the unobserved

value. One may argue that even the MCAR assumption might be reasonable in this case. On the other hand, the ten cases for projects without change history present a completely different missing data mechanism. Because the projects are older, they are likely to be different from newer projects in the analyzed sample. Data are missing because these projects are old (and presumably different) and, therefore, the MAR assumption does not apply. Consequently, the conclusions drawn from the analysis of the relationship between project tracking and project interval may not apply to old projects. We removed these projects from further consideration and narrowed conclusions to explicitly exclude them. For simplicity, we also excluded six observations where all tracking measures are missing. One can argue against such a decision, because these observations can still be used to make a more precise regression relationship between project size and project interval.

Many statistical packages deal with missing data by simply dropping the cases that have at least one value missing. Besides being inefficient (fewer observations are used for inference), such a technique may be biased unless the observations are MCAR. The MCAR assumption is rarely a reasonable assumption in practice.

Model based techniques where a statistical model is postulated for complete data provide transparency of assumptions, but other techniques are often simpler to apply in practice. Given that statistical software provides tools to deal with missing data using model based techniques (Schafer 1999, R Development Core Team 2005) we would recommend using them instead of the remaining techniques that have limited theoretical justification or require unrealistic assumptions. For completeness, we briefly describe most of traditional techniques as well. The goal of traditional techniques is to produce the sample mean or the covariance matrix to be used for regression, analysis of variance, or simply to calculate correlations. All traditional methods produce correct results under the MCAR assumption.

For more in-depth understanding of the statistical approaches Little & Rubin (1987) summarize statistical models for missing data and Schafer (1997) describes more recent results. Rubin (1987) investigates sampling survey issues. Little & Rubin (1989) and Schafer & Olsen (1998) provide examples with advice for practitioners. Roth (1994) provides a broad review of missing data technique application in many fields.

Various missing data techniques have been evaluated in the software engineering context of cost estimation. Strike, Emam & Madhavji (2001) evaluate listwise deletion, mean imputation, and eight different types of hot-deck imputation and find them to have small biases and high precision. This suggests that the simplest technique, listwise deletion, is a reasonable choice. However, it did not have the minimal bias and highest precision obtained by hot-deck imputation. Myrtveit,

Stensrud & Olsson's (2001) evaluate listwise deletion, mean imputation, similar response pattern imputation, and full information maximum likelihood (FIML) missing data techniques in the context of software cost modeling. They found bias for non-MCAR data in all but FIML technique and found that listwise deletion performed comparably to the remaining two techniques except in cases where listwise deletion data set was too small to fit a meaningful model. k-Nearest Neighbor Imputation is evaluated by simulating missing data in (Jönsson & Wohlin 2004). Authors's find the method to be adequate and recommend to use k equal to the square root of the number of complete cases. More recently, Twala, Cartwright & Shepperd (2006) compare seven missing data techniques using eight datasets and find listwise deletion to be the least efficient and multiple imputation to be the most accurate.

In the following sections we consider several broad classes of missing data techniques. Section 1.3.1 considers methods that remove cases with missing values. Ways to fill in missing values are considered in Section 1.3.2. Section 1.3.3 describes techniques that generate multiple complete datasets, each to be analyzed using traditional complete data methods. Results from these analyses are then combined using special rules. We exemplify some of these methods in Section 1.3.4

1.3.1 Deletion techniques

Deletion techniques remove some of the cases in order to compute the mean vector and the covariance matrix. *Casewise deletion*, *complete case*, or *listwise deletion* method is the simplest technique where all cases missing at least one observation are removed. This approach is applicable only when a small fraction of observations is discarded. If deleted cases do not represent a random sample from the entire population, the inference will be biased. Also, fewer cases result in less efficient inference.

In our example the *complete case* method loses 18 cases (around 34 percent of the 52 cases that we consider). Table 1.1 shows output from the multiple regression model in Equation (1.1).

Multiple regression shows that the project size is an important predictor of the interval but none of the process coefficients are significant at the ten percent level (although a five percent level is more commonly used, we chose to use a ten percent level that is more suitable for the small sample size of our example and, more importantly, to illustrate the differences among missing data methods). It is not too surprising, since more than a third of the observations were removed from

Table 1.1: Multiple regression for the *complete case analysis.*

Variable	Value	Std. Error	t value	$Pr(>\|t\|)$
Intercept	3.1060	5.2150	0.5956	0.5561
sqrt(size)	0.4189	0.1429	2.9315	0.0065
Tracking1	0.9025	0.9885	0.9130	0.3688
Tracking2	0.5363	1.2332	0.4349	0.6669
Tracking3	0.7186	1.1033	0.6513	0.5200

the analysis.

Pairwise deletion or *available case* method retains all non missing cases for each pair of variables. We need at least three variables for this approach to be different from listwise deletion. For example, consider the simplest example where the first of three variable is missing in the first case and the remaining cases are complete. Then, the sample covariance matrix would use all cases for the submatrix representing sample covariances of the second and third variables. The entry representing the sample variance of the first variable and sample covariances between the first and the remaining variables would use only complete cases. More generally, the sample covariance matrix is:

$$s_{jk} = \frac{\sum_{jk} R_{ik} R_{ij} (y_j - \overline{y}_j^k)(y_k - \overline{y}_k^j)}{\sum_i R_{ij} R_{ik} - 1},$$

where $\overline{y}_j^k = \sum_i R_{ij} R_{ik} y_{ij} / \sum_i R_{ij} R_{ik}$ and R_{ij} and R_{ik} are indicators of missing values as defined in Equation 1.2. Although such method uses more observations, it may lead to a covariance matrix that is not positive-definite (positive-definite matrix has positive eigenvalues) and unsuitable for further analysis, i.e., multiple regression.

1.3.2 Imputation techniques

The substitution or imputation techniques fill (impute) the values that are missing. Any standard analysis may then be done on the complete dataset. Many such techniques would typically provide underestimated standard errors.

The simplest substitution technique fills in the average value over available cases (*mean substitution*). This underestimates variances and covariances in MCAR

case and is likely to introduce bias otherwise. Smaller variances may reduce p-values and, therefore, may provide false impressions about the importance of some predictors. Table 1.2 shows results using *mean substitution*. Table shows that the project size is an important predictor of the interval and that the third dimension of tracking measure (level of agreement by all affected parties to the changes in the software commitments) might increase the interval. The coefficient is significant at 10 percent level.

Table 1.2: Results for the *mean substitution analysis.*

Variable	Value	Std. Error	t value	$Pr(>\|t\|)$
Intercept	3.1611	2.8054	1.1268	0.2656
sqrt(size)	0.3904	0.1134	3.4437	0.0012
Tracking1	-0.0871	0.5903	-0.1475	0.8834
Tracking2	0.8557	0.7339	1.1660	0.2495
Tracking3	1.4568	0.7678	1.8975	0.0639

Regression substitution uses multiple linear regression to impute missing values. The regression is done on complete cases. The resulting prediction equation is used for each missing case. *Regression substitution* underestimates the variances less than *mean substitution*. A stochastic variation of *regression substitution* replaces a missing value by the value predicted by regression plus a regression residual from a randomly chosen complete case.

Table 1.3 shows results based on a basic liner regression substitution. For our example the results are similar to mean substitution.

Table 1.3: Results for the *regression substitution* analysis.

Variable	Value	Std. Error	t value	$Pr(>\|t\|)$
Intercept	3.5627	3.3068	1.0774	0.2868
sqrt(Size)	0.3889	0.1242	3.1321	0.0030
Tracking1	0.0339	0.8811	0.0385	0.9695
Tracking2	0.6011	1.0760	0.5586	0.5791
Tracking3	1.5250	0.8518	1.7904	0.0798

Other substitution methods include *group mean substitution* that calculates means over groups of cases known to have homogeneous values within the group. A variation of group mean substitution when the group size is one is called *hot-deck* imputation. In *hot-deck* imputation for each case that has a missing value, a similar case is chosen at random. The missing value is then substituted using the value obtained from that case. Similarity may be measured using a Euclidean distance function for numeric variables that are most correlated with the variable that has a missing value.

The following two reasons prevent us from recommending simple deletion and imputation methods when a substantial proportion of cases (more than 10 percent) are missing:

1. it is not clear when they do not work;
2. they give incorrect precision estimates making them unsuitable for interval estimation and hypothesis testing.

As the percentage of missing data increases to higher levels, the assumptions and techniques have a more significant impact on results. Consequently, it becomes very important to use a model based technique with a carefully chosen model.

While there is no consensus among all experts about what techniques should be recommended, a fairly detailed set of recommendations is presented in (Roth 1994, Little & Hyonggin 2003), where factors such as proportion of missing data and the type of missing data (MCAR, MAR, NMAR) are considered. Roth (1994) recommends using the simplest techniques, such as pairwise deletion, in the MCAR case and model based techniques when the MAR assumption does not hold or when the percent of missing data exceeds 15 percent. Because we doubt the validity of the MCAR assumption in most practical cases we do not recommend using techniques that rely on it unless the percent of missing data is small.

1.3.3 Multiple imputation

Multiple imputation (MI) is a model based technique where a statistical model is postulated for complete data. A multivariate normal model is typically used for continuous data and a log-linear model is used for categorical data. In MI each missing value is replaced (imputed) by $m > 1$ plausible values drawn from their predictive distribution. Consequently, instead of one data table with missing values we get m complete tables. After doing identical analyses on each of the

tables the results are combined using simple rules to produce the estimates and standard errors that reflect uncertainty introduced by the missing data.

The possibility of doing an arbitrary statistical analysis for each complete data set and then combining estimates, standard deviations, and p-values allows the analyst to use a complete data technique that is the most appropriate for their problem. In our example we chose to use multiple linear regression.

The attractiveness of the MI technique lies in the ability to use any standard statistical package on the imputed datasets. Only a few (3-5) imputations are needed to produce quite accurate results (Schafer & Olsen 1998). Software to produce the imputed tables is available from several sources, most notably from (Schafer 1999, R Development Core Team 2005). We do not describe the technical details on how the imputations are performed because it is beyond the scope of this presentation and the analyst can use any MI package to perform this step.

After the m MI tables are produced, each table may be analyzed by any statistical package. To combine the results of m analyses the following rules are used (Rubin 1987). Denote the quantities of interest produced by the analyses as $P_1, \ldots, P_m$ and their estimated variances as $S_1, \ldots, S_m$.

- The overall estimate for P is an average value of P_i's: $\hat{P} = \sum_i P_i/m$;
- The overall estimate for S is $\hat{S} = \sum_i S_i/m + \frac{m+1}{m(m-1)} \sum_i (\hat{P} - P_i)^2$;

A rough confidence interval for P is $\hat{P} \pm 2\sqrt{\hat{S}}$. This inference is based on a t distribution and is derived under the assumption that complete data have an infinite number of degrees of freedom. A refinement of the rules for small datasets is presented in (Barnard & Rubin 1999). There $\hat{P}$ has a t distribution with variance $\hat{S}$ and degrees of freedom given by a fairly involved formula:

$$\left(\frac{1}{\nu} + \frac{1}{\hat{\nu}}\right)^{-1},$$

where $\nu = (m-1)/\gamma^2$, $\hat{\nu} = n\frac{n+1}{n+3}(1-\gamma)$, n represents degrees of freedom for complete data, and

$$\gamma = \frac{1}{\frac{k(m-1)\sum_i S_i}{(m+1)\sum_i (\hat{P}-P_i)^2} + k}$$

Sometimes the inference is performed on multiple quantities simultaneously, for example, if we want to compare two nested multiple regression models, where

the more general model has one or more extra parameters that are equal to zero in the simpler model. The rules for combining MI results in such a case are quite complicated, (see, e.g., pp. 112–118 Schafer 1997), however, the MI software (Schafer 1999) implements required calculations.

1.3.4 Example

We used the *norm* package (Schafer 1999) (also available as packages (Novo 2002) for R system (R Development Core Team 2005)) for Windows 95/98/NT platform to generate 5 imputations and ran multiple linear regression on each imputed data table. The estimates and standard errors from the regression were combined using multiple imputation rules. The *norm* package does not perform multiple regression, but it provides the functionality to combine the results from multiple regression analyses. We used this feature and the result is presented in Table 1.4. The coefficients are not much different from the regression imputation, although the third tracking dimension is now barely significant at the ten percent level.

Table 1.4: results of *multiple imputation* analysis.

Variable	Value	Std. Error	t value	$Pr(>\|t\|)$
Intercept	3.75	3.686	1.02	0.31
sqrt(Size)	0.39	0.126	3.12	0.002
Tracking1	0.01	0.787	0.02	0.985
Tracking2	0.56	1.114	0.51	0.614
Tracking3	1.51	0.917	1.65	0.099

In most practical situations with a medium percentage of missing data there will be relatively small difference between the results obtained using different missing data methods (except for the *complete case* method), as happens to be the case in our example. However, in many examples (like this one), where the conclusions are based on p-values that are close to the chosen significance level, the use of MI is essential. In particular, the *mean substitution* method was significant at 0.07 level, but the MI method was not. If we, hypothetically, assume a world where results are judged to be significant at 0.07 significance level (instead of our own world, where the 0.05 significance level is most common), we would have reached different conclusions using different methods.

The example reiterates the fact that the standard deviation is underestimated in imputation methods and, therefore, the significance values are inflated. Although this example does not show large biases introduced by non MI methods, in general it may be a serious issue. The example also illustrates the lack of efficiency of the *complete case* method in line with the studies mentioned above.

1.4 Other types of unavailable data

Software engineering has its own domain-specific types of missing data that are not present in the general statistical treatment. Here we briefly present specific cases of missing data in software artifacts. The first example deals with missing information on software change purpose, and the second example deals with missing information on software change effort.

1.4.1 Determining change purpose

Three primary driving forces in the evolution of software are: *adaptive* changes introduce new functionality, *corrective* changes eliminate faults, and *perfective* changes restructure code in order to improve understanding and simplify future changes (Swanson 1976, An, Gustafson & Melton 1987). Models of software evolution must take into account the significant differences in purpose and implementation of the three types of changes (Graves, Karr, Marron & Siy 2000, Atkins, Ball, Graves & Mockus 1999). However, few change history databases record such information directly. Even if a record exists, it is rarely consistent over time or across organizations. Fortunately, change history databases usually record a short description of the purpose for the change at the maintenance request (MR) or lower level. Such description or abstract is provided by developers who implement the change.

Work in (Mockus & Votta 1997) used textual analysis of MR abstracts to impute adaptive, corrective, or perfective labels to the changes. It classified MRs as adaptive, corrective, or perfective depending on which key words appear in these change abstracts. The classification scheme was able to tag around 85% of all MRs.

1.4.2 Estimating change effort

A particularly important quantity related to software is the cost of making changes. Therefore, it is of great interest to understand which factors have historically had strong effects on this cost, which could be approximated by the amount of time developers spend working on the change.

When performing historical studies of cost necessary to make a change, it is important to study changes at a fine level (MRs as opposed to releases). Studying larger units of change, such as releases, may make it impossible to separate the effects of important factors. For example, software releases typically contain a mixture of several types of changes, including new code and bug fixes. Consequently, the relative effort for the different types of changes can not be estimated at the release level. Also, larger change units may involve multiple developers and distinct parts of the code, making it difficult to estimate developer effects.

Measurements of change effort are not recorded in a typical software production environment. Graves & Mockus (1998) describe an iterative imputation algorithm that, in effect, divides a developer's monthly effort across all changes worked on in that month. The algorithm uses several measurements on each change including the size and type of a change. Both measures are related to the amount of effort required to make the change. The effort estimation tools provide valuable cost driver data that could be used in planning and in making decisions on how to reduce expenses in software development.

1.5 Summary

It should be noted that the quality of collected data will have more influence on the analysis results and the success of a study than a choice of method to deal with missing values. In particular, a successful data collection might result in few or no missing values.

In many realistic scenarios the data quality is low, and some values are missing. In such cases, the first step should be to determine the mechanism by which the data are missing and add observations that may explain why the values are missing. This would make the MAR assumption more plausible. For MAR (and MCAR) data, *multiple imputation* mitigates the effects of missing values. Other research and our case study have shown not only the importance of applying a missing data technique such as imputation, but also the importance of carrying out multiple imputation. In our case study we find that different conclusions may

be reached depending on the particular method chosen to handle missing data. This demonstrates that the selection of a proper method to handle missing data is not simply a formal exercise, but it may, in certain circumstances, affect the outcome of an empirical study.

Bibliography

Albrecht, A. J. & GaffneyJr., J. E. (1983), 'Software function, source lines of code, and development effort prediction: a software science validation', *IEEE Trans. Software Eng.* **9**(6), 639–648.

An, K. H., Gustafson, D. A. & Melton, A. C. (1987), A model for software maintenance, *in* 'Proceedings of the Conference in Software Maintenance', Austin, Texas, pp. 57–62.

Atkins, D., Ball, T., Graves, T. & Mockus, A. (1999), Using version control data to evaluate the effectiveness of software tools, *in* '1999 International Conference on Software Engineering', ACM Press, pp. 324–333.

Barnard, J. & Rubin, D. B. (1999), 'Small sample degrees of freedom with multiple imputation', *Biometrika* **86**(4).

Chidamber, S. R. & Kemerer, C. F. (1994), 'A metrics suite for object oriented design', *IEEE Trans. Software Eng.* **20**(6), 476–493.

Fleming, T. H. & Harrington, D. (1984), 'Nonparametric estimation of the survival distribution in censored data', *Comm. in Statistics* **13**, 2469–86.

Goldenson, D. R., Gopal, A. & Mukhopadhyay, T. (1999), Determinants of success in software measurement programs, *in* 'Sixth International Symoposium on Software Metrics', IEEE Computer Society, pp. 10–21.

Graves, T. L., Karr, A. F., Marron, J. S. & Siy, H. P. (2000), 'Predicting fault incidence using software change history', *IEEE Transactions on Software Engineering* **26**(7), 653–661.

Graves, T. L. & Mockus, A. (1998), Inferring change effort from configuration management databases, *in* 'Metrics 98: Fifth International Symposium on Software Metrics', Bethesda, Maryland, pp. 267–273.

Halstead, M. H. (1977), *Elements of Software Science*, Elsevier North-Holland.

Herbsleb, J. D. & Grinter, R. (1998), Conceptual simplicity meets organizational complexity: Case study of a corporate metrics program, *in* '20th International Conference on Software Engineering', IEEE Computer Society, pp. 271–280.

Herbsleb, J. D., Krishnan, M., Mockus, A., Siy, H. P. & Tucker, G. T. (2000), Lessons from ten years of software factory experience, Technical report, Bell Laboratories.

Jönsson, P. & Wohlin, C. (2004), An evaluation of k-nearest neighbour imputation using likert data, *in* 'Proc. of the 10th Int. Symp. on Software Metrics', pp. 108–118.

Kaplan, E. & Meyer, P. (1958), 'Non-paramentric estimation from incomplete observations', *J Am Stat Assoc* pp. 457–481.

Kim, J. & Curry, J. (1977), 'The treatment of missing data in multivariate analysis', *Social Methods and Research* **6**, 215–240.

Little, R. & Hyonggin, A. (2003), Robust likelihood-based analysis of multivariate data with missing values, Technical Report Working Paper 5, The University of Michigan Department of Biostatistics Working Paper Series.
`http://www.bepress.com/umichbiostat/paper5`

Little, R. J. A. (1988), 'A test of missing completely at random for multivariate data with missing values', *Journal of the American Statistical Association* **83**(404), 1198–1202.

Little, R. J. A. & Rubin, D. B. (1987), *Statistical Analysis with Missing Data*, Willey Series in Probability and Mathematical Statistics, John Willey & Sons.

Little, R. J. A. & Rubin, D. B. (1989), 'The analysis of social science data with missing values', *Sociological Methods and Research* **18**(2), 292–326.

McCabe, T. (1976), 'A complexity measure', *IEEE Transactions on Software Engineering* **2**(4), 308–320.

Mockus, A. (2006), Empirical estimates of software availability of deployed systems, *in* '2006 International Symposium on Empirical Software Engineering', ACM Press, Rio de Janeiro, Brazil, pp. 222–231.

Mockus, A. (2007), Software support tools and experimental work, *in* V. Basili & et al, eds, 'Empirical Software Engineering Issues: LNCS 4336:', Springer, p. to appear.

Mockus, A. & Votta, L. G. (1997), Identifying reasons for software changes using historic databases, Technical Report BL0113590-980410-04, Bell Laboratories.

Myrtveit, I., Stensrud, E. & Olsson, U. (2001), 'Analyzing data sets with missing data: An empirical evaluation of imputation methods and likelihood-based methods', *IEEE Transactions on Software Engineering* **27**(11), 1999–1013.

Novo, A. (2002), 'Analysis of multivariate normal datasets with missing values'. Ported to R by Alvaro A. Novo. Original by J.L. Schafer.

R Development Core Team (2005), *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
`http://www.R-project.org`

Roth, P. L. (1994), 'Missing data: A conceptual review for applied psychologist', *Personel Psychology* **47**, 537–560.

Rubin, D. B. (1987), *Multiple Imputation for Nonresponse in Surveys*, John Willey & Sons.

Schafer, J. L. (1997), *Analysis of Incomplete Data*, Monograph on Statistics ans Applied Probability, Chapman & Hall.

Schafer, J. L. & Olsen, M. K. (1998), 'Multiple imputation for multivariate missing data problems', *Multivariate Behavioural Research* **33**(4), 545–571.

Schafer, J. S. (1999), 'Software for multiple imputation'.
`http://www.stat.psu.edu/~jls/misoftwa.html`

Strike, K., Emam, K. E. & Madhavji, N. (2001), 'Software cost estimation with incomplete data', *IEEE Transactions on Software Engineering* **27**(10), 890–908.

Swanson, E. B. (1976), The dimensions of maintenance, *in* 'Proc. 2nd Conf. on Software Engineering', San Francisco, pp. 492–497.

Twala, B., Cartwright, M. & Shepperd, M. (2006), Ensemble of missing data techniques to improve software prediction accuracy, *in* 'ICSE'06', ACM, Shanghai, China, pp. 909–912.

Weisberg, S. (1985), *Applied Linear Regression, 2nd Edition*, John Wiley & Sons, USA.