

# Two Case Studies in Measuring Software Maintenance Effort\*

Frank Niessink and Hans van Vliet

Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam

De Boelelaan 1081, 1081 HV, Amsterdam, The Netherlands

E-mail: {F.Niessink, J.C.van.Vliet}@cs.vu.nl

## Abstract

*In this paper we present the results of two measurement programs, that were aimed at investigating possible cost drivers for software maintenance. The two measurement programs were implemented in the software maintenance departments of two different organizations. Both programs were set up in roughly the same way. We use standard statistical techniques – principal component analysis and multiple regression analysis – to analyse the datasets. Surprisingly, with one of the datasets we are able to explain a fair amount of variance in the effort, while with the other dataset we can explain much less. From a closer inspection of the different environments we conjecture that the existence of a consistently applied process is an important prerequisite for a successful measurement program. In addition, if the process exists in multiple variants, it is important to know which variant is applied when.*

## Keywords

Software maintenance, software measurement, measurement program, maintenance effort, maintenance cost drivers, software maintenance process, empirical study.

## 1. Introduction

One of the many difficult aspects of software engineering, including software maintenance, is the estimation of effort needed to build, correct or enhance a software system. Organizations that wish to improve their planning need to investigate which factors influence their maintenance or development process most. Generally, in immature organizations, little is known about these factors. In these cases, measuring *possible* cost drivers is the only way to find out which factors influence effort. However, measurement can

be quite expensive, so measuring everything in sight is probably not an adequate solution. The question is what to measure, and what not to measure.

In this paper we investigate two measurement programs that were both aimed at identifying possible cost drivers for software maintenance change requests. The measurement programs were implemented in the software maintenance departments of two different organizations. Both these organizations were immature in that there was little or no a priori qualitative or quantitative data on maintenance cost drivers.

Both programs were set up in roughly the same way. A principal component analysis of the datasets reveals that the main factors present in the two datasets are very similar. Surprisingly though, we are able to explain a fair amount of variance in maintenance effort for one of these datasets, while we utterly fail to do so for the other dataset. A closer inspection of the different maintenance environments shows that neither organization applies *one* defined, maintenance process. This turns out to be no problem, as long as the process is applied consistently and we know which variant of the maintenance process is executed when. We conjecture that these are major prerequisites for a successful measurement program.

This paper is structured as follows. In the next section, we give an overview of related research. Next, in section 3 we describe the implementation of the two measurement programs. In section 4 we present our data analysis and the results thereof. In section 5 our conclusions and indications for future work are given.

## 2. Related research

As described in the previous section, we are interested in improving insight in the cost drivers for maintenance change requests. If we turn to empirical research on software maintenance to identify potential cost drivers, we have to discriminate between different levels on which cost drivers can be found. We call these the macro, meso and micro level of software maintenance:

---

\*Published in the proceedings of the International Conference on Software Maintenance, Bethesda, Maryland, USA, November 16-20, 1998, pp. 76-85.

**Macro level** This level is comprised of the total life-cycle costs or total maintenance costs of a software system. An example of research on this level is Gefen and Schneberger [4], who look at the distribution of different types of software modifications during the life cycle of a software system.

**Meso level** Costs of maintenance releases or large enhancements. For example, Basili et al. [2] report on their efforts to develop a model to estimate the cost of maintenance releases. A distinction is made between different maintenance change types and maintenance activities. For a subset of 21 enhancement releases, a model is fitted that predicts release effort from source lines of code added, changed or deleted.

**Micro level** The costs of individual maintenance tasks.

Focusing on the third category, we see that there is a large number of different cost drivers being investigated. We give a few examples:

**Maintenance type** Most research on software maintenance tasks takes the type of the maintenance task into account. For example, Jørgensen [7] tests several hypotheses on a dataset of 124 maintenance tasks from a large Norwegian company. Among his findings is that corrective maintenance tasks cost four to five times less effort than adaptive, perfective or preventive tasks. Similar results are reported by Abran and Nguyenkim [1], who looked at the distribution of effort across corrective, adaptive, perfective and user support activities in a large Canadian financial institution. Their findings suggest a difference between corrective maintenance tasks on the one hand and adaptive and perfective on the other. However, the difference is not as large as in [7].

**Complexity** Jørgensen [7] also reports on the difference between tasks considered to be of high and low complexity. The 12% tasks with a high complexity took 25% of the total effort, while the 48% tasks which were considered to be of low complexity, costed 20% of the total effort.

**Requirement changes** Henry et al. [6] present examples from a measurement-based maintenance process set up at Lockheed-Martin. They show how the number of requirements changes that occur during the maintenance process can be used to improve effort estimates.

**Size** In [8], we examined function point data concerning the maintenance of a large administrative information system. We found that the size of the software component changed was more important than the function point model used implied. Adjusting the model to our findings improved the model considerably.

**And more...** Many more maintenance task characteristics have been investigated. For example, Evanco [3] identifies three determinants for fault correction effort: fault locality, characteristics of the defective software components and the cumulative changes made to the software. Jørgensen divides maintenance tasks into several categories depending on the type of change that was made to the code. This explains some of the effort, for example, the 20% of the tasks that were mainly concerned with introduction or deletion of modules took 40% of the total effort.

Although a large number of cost drivers has been investigated, there does not seem to be a consensus on which characteristics of maintenance tasks are the most important cost drivers.

### 3. Two Measurement Programs

In section 3.1, we give a characterization of the two organizations<sup>1</sup>. In section 3.2, we describe the different steps that were taken to implement the measurement programs. Section 3.3 presents an overview of the data that were gathered.

#### 3.1. The organizations

##### Organization A

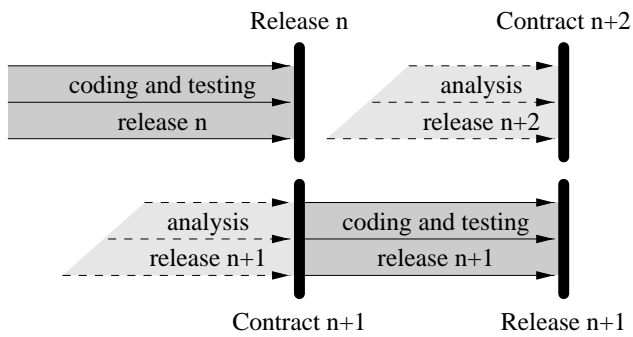
The first organization, which we will call organization A, is the IT department of a large organization, responsible for carrying out part of the Dutch social security system. As of the beginning of 1996, the organization has been split into a non-profit public body and a private for-profit organization – part of which is the IT department. In the future, the IT department will have to compete with third parties. However, at the moment this is not yet the case and the majority of the customers of the IT department are still departments from the non-profit sibling organization.

The IT department consists of several units, one of which contains the so-called product teams. Each product team develops, maintains and supports multiple systems for one (department of the) customer. Each team is further divided into different groups that each maintain several of the customer's information systems. Our measurement program was introduced into three of the eight product teams.

Each product team consists of about 20 to 30 engineers. Each group in a product team is staffed with between one and five people. Contacts with the customer are handled by the team managers and group leaders. The customer submits change requests that are analyzed by the responsible

---

<sup>1</sup>The two organizations described here are referred to as organization C and D in [9].



**Figure 1. Maintenance releases at organization A**

team manager or group leader. The change requests are then implemented and delivered in the next release, as displayed in figure 1. Note that the contract for release  $n + 1$  can only be finalized when all of its changes have been analyzed and agreed upon. This point in time roughly coincides with the delivery of release  $n$ . Most systems have three or four releases per year, with between one and ten change requests per release.

Officially, change requests undergo five distinct phases before delivery, see figure 2. However, during the implementation of the measurement program it turned out that the preparation and functional design are usually done by the same person, the group leader. The same goes for the technical design and building, which are usually done by the same engineer. Therefore, group leaders and engineers often do not know how much time they have spent on each of these phases, which makes it hard to distinguish between them. In our analysis, we will therefore use three phases, indicated by a shaded background in figure 2: (1) *analysis*, which consists of preparation and functional design, (2) *coding*, which is the sum of technical design and build, and (3) testing.

In one of the three teams the testing of change requests was not done by the engineers in the team, but was outsourced to a separate test team.

### Organization B

Organization B is the IT department of a large Dutch industrial organization. The IT department consists of several units, one of which – the applications unit – is responsible for the development and maintenance of the administrative and process-control systems that the organization uses. The measurements took place at two of the three subunits of the applications unit. Each subunit is staffed with 20 to 30 engineers.

For each of the information systems, there is an interme-

diary between the client and the maintenance department, see figure 3. This intermediary is located at the client site. He or she is responsible for phrasing the change requests. The intermediary is in direct contact with the programmer at the IT department who maintains the systems of his department. The amount of analysis and design done by the intermediary varies per system: some intermediaries change the functional documentation themselves, others give an informal description of the change and leave it up to the engineer to change the functional documentation. We only collected the effort spent by the engineers, not the intermediaries, as indicated by the shaded box in figure 3.

Budgets for maintenance are allocated per system per year. Change requests are implemented, tested and delivered one by one, as opposed to being grouped in releases.

### 3.2. Implementing the measurement programs

Each measurement program was set up by a (different) graduate student. The goal of both measurement programs was to gain insight into maintenance cost drivers, in order to support the estimation and planning of software maintenance tasks. The measurement programs had a duration of about 7 months.

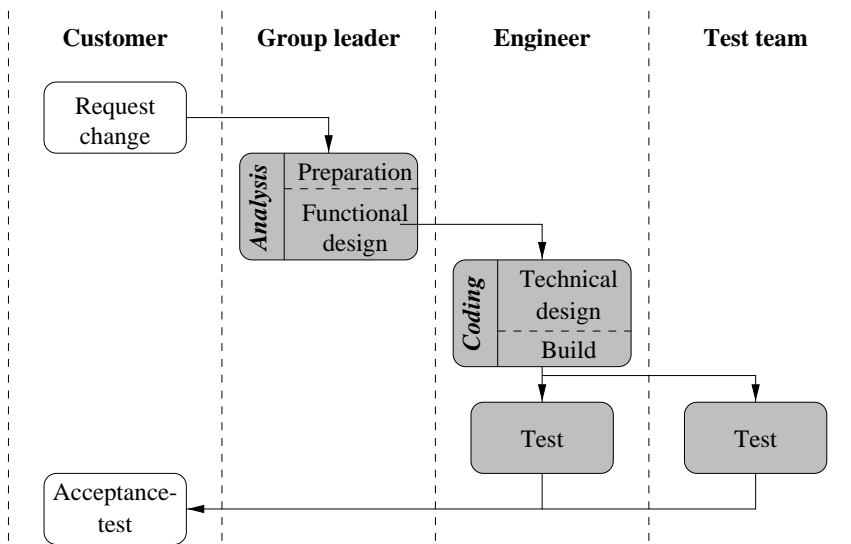
Both students used the same steps to implement the program:

1. Model the maintenance processes.
2. Determine likely maintenance cost drivers.
3. Develop forms to collect the data.
4. Collect the data by ‘walking around’.
5. Use flyers to provide feedback to the engineers.
6. Analyse the data.
7. Present the conclusions to the engineers and management.

During the first step, the students drew up a process model of the maintenance process used in the organizations. They based their model mostly on interviews with managers and engineers. In neither organization a formal standard process existed. The students were able to derive a de facto standard process that was more or less used by all the engineers. However, different groups, even within the same unit or team, would execute different variants of the de facto standard process.

In the second step likely drivers of maintenance costs were determined, based on literature research and interviews with managers and engineers. The upper part of table 1 shows the cost drivers mentioned (indicated by a  $\checkmark$ ) by engineers and management.

It is surprising that size attributes were not mentioned as possible cost drivers. Because both organizations use



**Figure 2. Maintenance process organization A**

Cost drivers mentioned . . . . in organization:	A	B
Type of maintenance activity (corrective, adaptive, etc.)	✓	✓
Changing requirements		✓
Work needed to convert data		✓
Changed use of the database	✓	✓
User interface change	✓	
Code structuredness, readability, and quality		✓
Experience of the engineer with the code		✓
Kind of database used		✓
Relationship with other applications		✓
Relationship with other change requests	✓	
Readability, completeness, clarity, and structure of the documentation		✓
Availability of testsets	✓	✓
Tests performed	✓	
Complexity of the change	✓	
Size of the code to be changed		
Size of the change		
Application characteristics		

**Table 1. Candidate cost drivers**

source code version control systems, and thus the source code and changes to the code are easily available, it was decided to also collect source code size metrics.

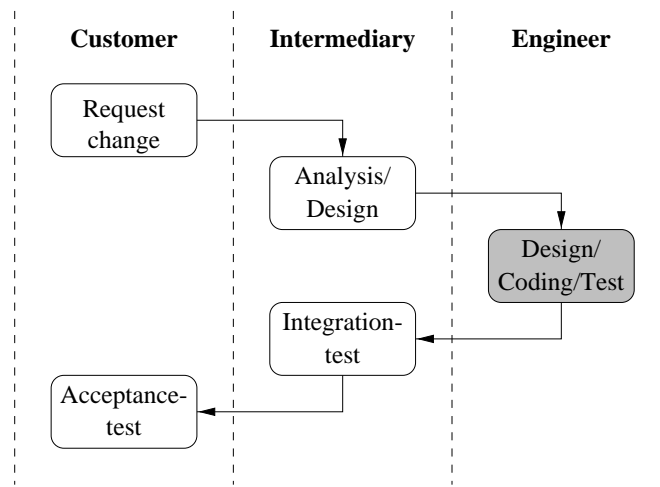
It was decided to not gather application characteristics, such as e.g. programming language, size of the application, number of users, etc. One reason is that application factors were not mentioned as possible cost drivers in the interviews. Moreover, the number of change requests per

application would be relatively small – between one and ten. This makes it difficult to accurately compare the influence of application characteristics on the effort to implement changes.

Using the possible cost drivers, the students developed forms to collect the data. In both organizations, engineers register their hours using an effort registration system. However, in organization A, the effort is not registered per change request, but per release. So, in organization A, the hours planned and spent per change request needed to be registered on the forms. In organization B, this was unnecessary, since the hours per change request were available from the effort registration system. In both organizations, code metrics were gathered after the changes were completed, using the version control systems.

Next, the forms were handed out to the relevant managers and engineers in the organization to be filled in. Knowing exactly who was working on which change request, the student regularly visited these persons to see whether the change requests were ready. This guaranteed the forms would be filled in and collected timely.

During both projects, feedback was given using flyers with information about the status of the project and intermediate results. At the end of the projects both students performed a preliminary data analysis, using statistical tools, and presented those results to the people involved in the measurement programs.



**Figure 3. Maintenance process organization B**

### 3.3. The datasets

#### Organization A

The dataset of organization A contains information about 84 change requests concerning 13 different information systems. Of these 13 systems, 12 are written in Cobol, and one system both in Cobol and with Powerbuilder. For 18 change requests, only the first phase (analysis) was performed, i.e. the actual implementation was either canceled or deferred to a later release. For 46 of the 57 change requests that were both completed and concerned Cobol only, code metrics were gathered from the version control system. For each change request the following attributes were gathered by use of the forms:

**Functional Design** Does the functional design remain unchanged, is it changed, or is it to be newly designed (functional design).

**Database** This attribute indicates whether the *structure* of the used database has to be changed due to the change. Since this happened only twice, we did not include this attribute in our analysis.

**Complexity** The planners were asked to assess the complexity of the change on a five-point-scale, ranging from very simple to very difficult (complexity).

**Screens** The number of new screens and the number of screens changed due to the change request (screens changed and screens new).

**Lists** Lists are output of batch programs that provide information to the user about the processing of the batch jobs. Like screens provide the user interface for interactive programs, lists provide the ‘user interface’ for batch programs.

A list provides information about the number of transactions processed, an overview of the transactions processed, an overview of transactions that failed, an overview of database tables used, etc. We measured the increase of the number of lists (lists new) and the number of lists that were changed (lists changed).

**Files** Files are all configuration items in the version control system, excluding documentation. Examples are: COBOL program files, COBOL module files, job control files, record definitions and screen definitions. COBOL programs are COBOL files that result in an executable program. Programs implement the main flow-of-control of the system. COBOL modules contain code that is used (‘called’) by programs or other modules.

Information about the number of files that were to be deleted, changed or added in the course of a change request (files deleted, files new, files changed) was collected. We also separately asked for the number of programs and modules (programs deleted, programs new, programs changed, modules deleted, modules new and modules changed). Note that in general programs changed + modules changed < files changed, because of other types of files that are changed also. Examples are job control files and record definitions.

Using the version control system, we also measured the total size (in lines of code) of the changed files (files loc), and separately of the changed programs and changed modules (programs loc and modules loc). We did the same for new files (files new loc, programs new loc and modules new loc). We also measured the size of the change by comparing the old and new versions of files using the UNIX program `diff`. This results

in two measures: the number of lines added and the number of lines deleted. Again, these measures were taken for all files together (files loc added and files loc deleted) and separately for programs (programs loc added and programs loc deleted) and modules (modules loc added and modules loc deleted).

**Tests** The forms also contained two questions about testing, but in the course of the measurement program it turned out that these questions were misunderstood and answered inconsistently. These data were omitted from the dataset.

Variable	Mean	Std. dev.	Min.	Max.
effort total	71	56	7	260
effort analysis	14	16	0	76
effort coding	44	45	0	192
effort test	22	23	0	81

**Table 2. Effort organization A in hours**

Table 2 shows the mean and standard deviation of the effort spent on the 57 change requests in organization A. There is one outlier in this dataset with respect to total effort. This change request took 472 hours, while the next biggest cost 260 hours and the average effort is 71 hours (excluding the outlier). The large amount of effort for this change requests seems to be caused by a large testing effort, while the other characteristics have normal values. We do not exclude this change request from our dataset; we only ignore it when analysing testing effort and total effort.

### Organization B

In organization B we have collected data on 63 changes that were applied to 9 different systems. Most of these systems are written completely or partially in COBOL. Of the 63 change requests, 5 changes concerned non-COBOL code. These change requests are not used in the analysis. For 52 of the remaining 58 changes, we have collected source code metrics. This dataset also contains one outlier: a change request that took 302 hours to complete. The next biggest change request took 130 hours, the average is 35 hours (excluding the outlier). Since we want our analysis to hold for the majority of the data, the outlier is removed from the dataset. This leaves us with 51 datapoints.

Variable	Mean	Std. dev.	Min.	Max.
effort total	35	30	0	130

**Table 3. Effort organization B in hours**

Table 3 shows the effort spent on change requests in organization B. Next to the total effort data, which was taken from the organization’s effort registration system, information was gathered on the hours spent on different phases of the maintenance process – design, coding and testing. However, the hours spent on each phase were to be estimated as a percentage of the total hours per change request. This did not result in correct data, not only because the percentages were estimated, but also because sometimes more than one person would work on a change request. Because the form was filled in by only one of these persons, we only know the effort distribution of that person. Hence, we do not further look at the effort data per phase.

For each change request, the following attributes were measured:

**Entities** The number of database entities used in the new version that were not used before the change (entities new) and the number of entities whose usage changed, i.e. the usage of one or more attributes changed (entities changed).

**Attributes** The number of attributes used in the new version of the software that were not used before the change (attributes new), plus the number of attributes no longer used after the change (attributes deleted).

Note that if entities or attributes are changed, i.e. the *usage* changes, this does not imply that the underlying database structure has changed.

**Files** The total number of changed COBOL files (files), and separately the number of changed modules (modules) and the number of changed COBOL programs (programs) are measured. In addition, the number of new programs or modules (new files) is measured. Note that as opposed to organization A, we only measured COBOL sources at organization B, so programs + modules = files.

Using the version control system, the length of changed files (loc), the amount of lines changed (loc added and loc deleted), and the length of the new files (loc new) were measured. Also, for each of these the number of lines changed, added, or new in the procedure division was counted, resulting in loc pd added, loc pd deleted, loc pd, and loc pd new.

**Requirements** Because the programmers were frequently confronted with changing customer requirements during the implementation of change requests, the number of such requirement changes was counted (new requirements).

**Data conversion** The number of temporary programs for conversion purposes (temporary programs).

The following attributes are constructed from attributes that were measured per file, as opposed to the other attributes

that were measured per change request. For example, we looked at each source file so see whether goto statements were used. If at least one of the source files that is changed contains goto statements, the variable goto is one, otherwise it is zero.

**Code quality** The quality of the source code was measured by two ordinal measures: does the source contains goto-statements (goto) and do all programs and modules have a good structure (structure).

**Program type** The program type was measured to distinguish between interactive programs and batch programs, because engineers consider changes in interactive programs more difficult than changes in batch programs. program type is batch (zero) if all of the changed modules or programs are only used for batch operations, otherwise the program type is interactive (one).

**Documentation** The quality of the documentation (documentation quality) is zero if one or more documents necessary for the change is of low quality, one otherwise.

**Experience** The experience of the programmer with the code to be changed (experience) is one if the programmer has much experience with all files to be changed, zero otherwise.

**Difficulty** The code to be changed is deemed to be difficult (difficulty is one) if at least one of the files to be changed is considered difficult.

Attributes	Factors		
	A1	A2	A3
programs loc added	0.86		
programs changed	0.85		
files loc added	0.82		
programs loc	0.76		
files changed	0.76		0.40
programs loc deleted	0.71		
files loc deleted	0.70		
files loc	0.63		
lists changed	0.59		
lists new	0.31		
files loc new		0.90	
programs loc new		0.84	
programs new		0.83	
files new		0.81	
modules new		0.64	0.33
functional design		0.54	
modules loc new		0.47	
modules loc added			0.87
modules changed			0.82
modules loc deleted			0.79
modules loc			0.57
complexity			0.45
screens changed			0.41
screens new			
Eigenvalue	5.32	4.03	3.46
Percentage of variance	22.2	16.8	14.4
Cumulative percentage	22.2	39.0	53.4

## 4. Data Analysis

In this section, we present the results of our analysis of the data gathered. Because we have a rather large number of attributes for both organizations, especially when compared to the number of observations, we use principal components analysis to determine the main underlying dimensions of the datasets. Using the constructed factors, we build regression models to attempt to explain the effort spent.

### 4.1. Principal components analysis

#### Organization A

The first step in principal components analysis is to decide on the appropriate number of factors for the dataset. Using the unrotated factor matrix and a plot of the eigenvalues of the factors (a so-called scree plot) we observe that the common variance in this dataset is best described by three factors. Table 4 shows the three factors, after varimax rotation. For readability, all factor loadings between  $-0.30$  and  $0.30$  have been omitted. Together these three constructed variables explain slightly more than 50% of the variance among

**Table 4. Rotated factor matrix for dataset A**

the attributes used. We see from table 4 that the factors can be easily interpreted:

- A1. The first factor can be interpreted as the amount of change that affects the flow-of-control. As noted before, programs implement the main flow-of-control. A change in a program file is therefore likely to affect other configuration items, such as job control files, as well. For the same reason, such changes can affect the processing information that batch programs return in different lists (lists changed). This factor is thus dominated by flow-of-control effects.
- A2. Factor 2 is the increase in the size of the system in terms of new files. The number of new files, new programs and new modules all load on this factor, as well as the size of the new files (files loc new, programs loc new and modules loc new). Also functional design has its highest loading on this factor. This is not surprising, since functional design was coded 0 for no change to the functional design, 1 for a change, and 2 for a

new functional design.

A3. The third factor reflects the amount of change in modules.

Two of these factors thus reflect the amount of change to existing code. Factor A1 can be roughly characterized as control flow change, and factor A3 can be labeled algorithm change. Factor A2 denotes the new code, and apparently no distinction as to the type of addition can be made.

### Organization B

Attributes	Factors		
	B1	B2	B3
files	0.87		
loc pd	0.85		-0.31
loc	0.85		-0.32
modules	0.68		
used-by	0.68		-0.33
programs	0.53		
entities changed	0.43		0.32
relationships	0.36		
goto statements	0.32		
difficulty	0.32		
entities new			
loc pd deleted		0.93	
loc deleted		0.93	
temporary programs		0.72	
loc pd added		0.71	-0.33
loc added		0.67	-0.35
new requirements		0.63	
structure		0.43	
documentation quality			
new files			0.85
loc new			0.83
loc pd new			0.82
experience			-0.64
testing			0.53
program type	0.37		-0.43
attributes new			0.42
attributes deleted			
Eigenvalue	5.70	3.71	3.27
Percentage of variance	21.1	13.8	12.1
Cumulative percentage	21.1	34.9	47.0

**Table 5. Rotated factor matrix for dataset B**

We also perform principal components analysis on dataset B. Using the unrotated factor matrix we observe that a number of three factors fits this dataset best. Table 5 shows the factor loading matrix after varimax rotation. Again, for

readability all factor loadings between  $-0.30$  and  $0.30$  have been left out.

The three factors explain nearly 50% of the variance among the used attributes. Using table 5, we see that interpretation of the three factors is again not too difficult:

B1. The first factor can be interpreted as the total size of the code components affected by the change.

B2. The second factor denotes the amount of change.

B3. We see that the third factor can be interpreted as the amount of code added.

Note that this dataset does not reveal a difference in the type of change (control-flow versus algorithmic). On the other hand, it does discriminate between the change itself and the system ‘slice’ affected by the change.

### 4.2. Regression analysis

The next step in our analysis is to investigate whether we can use the factors found in the principal components analysis to explain the effort spent on the change requests. We perform multivariate regression analysis, using the factors found in section 4.1. We use the beta coefficients of the independent variables to indicate the relative weight of each of the variables in the equation.

#### Organization A

Input variables for the stepwise regression analysis are the factors found in section 4.1, completed with two dummy variables to discriminate between the three teams. The results of the regression analysis is shown in table 6. For each dependent variable, the table shows which independent variables entered the regression formula. The numbers shown are the beta coefficients of the variables entered, which allow us to assess the relative importance of each of the independent variables entered in the equation.

If we look at the adjusted  $R^2$ 's, we see that the equation for the total effort explains 58% of the variance. Using the beta coefficients, we see that A1 is the most influential variable, followed by A2. The formulas for effort coding and effort test both explain about two-third of the variance. The main difference is that for effort test the team dummy variables (team A and team B) play the most important role. The beta coefficients are negative, because the average testing effort of these two teams is considerably lower than for the third team. This is explained by the fact that the third team outsources the testing to a separate test team, while the other two teams test the change requests themselves, see figure 2. If we omit factor A1 from the analysis, we see that the explained variance from the two dummy variables alone is almost 50%. Adding A1 increases the explained variance with about 20 percentage points. So, although the explained

Dependent variable	Independent variables					Adjusted $R^2$	Std. Err.
	A1	A2	A3	team A	team B		
effort total	0.56	0.41	0.34			0.58	37.1
effort analysis			0.46			0.20	13.6
effort coding	0.60		0.56			0.65	23.8
effort test	0.42			-0.67	-0.80	0.66	13.9
effort test <sup>a</sup>				-0.68	-0.78	0.48	17.1

<sup>a</sup>Without A1

**Table 6. Stepwise multivariate regression analysis for organization A**

variance for testing is as high as for coding, its source is rather different.

The explained variance for effort analysis is much lower than for the other equations. It seems that the factors that we have constructed have little influence on the effort needed to prepare the change request. This does suggest that it is important to look at the maintenance *process* carefully and select different metrics for each of the process steps.

### Organization B

Dependent variable	Independent vars.			Adjusted $R^2$	Std. Err.
	B1	B2	B3		
effort total	0.33			0.09	28.5

**Table 7. Stepwise multivariate regression analysis for organization B**

Using the factors found in section 4.1, we again perform stepwise regression analysis. The regression analysis does result in a formula, but the explained variance is not nearly as high as for the dataset of organization A. Only one variable enters the equation: B2 – the amount of code changed.

It is difficult to investigate this further, since we don't know how much of the effort in organization B was spent on design, coding and testing. As mentioned in section 3.3, the way in which we attempted to collect information about maintenance subtasks failed to deliver reliable data. Hence, we can only hypothesize on the reasons why the information gathered in this measurement program explains so little about the maintenance effort, as opposed to the results for organization A. We think there are two possible reasons for this difference:

- The maintenance process at organization B is quite dependent on the specific maintenance programmer and his relationship with the customer. The amount of analysis done by the programmer largely depends on

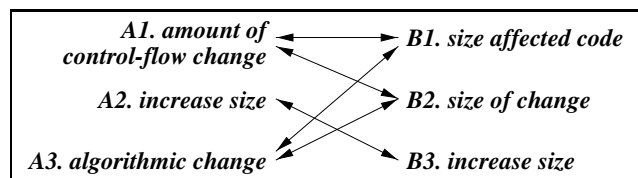
how much analysis the intermediary at the customer site has done.

- The effort data were taken from the effort administration system. We do not know – and had no control over – the accuracy with which these data were registered.

## 5. Conclusions

In this paper we have presented two measurement programs, their characteristics, the data gathered, and an analysis of the datasets. We have shown how common statistical techniques such as principal components analysis and regression analysis can be used to explore relationships between characteristics of change requests and the effort needed to implement them.

With the factors found in the dataset of organization A we were able to explain a fair amount of variance in the effort to implement change requests. This provides a useful starting point for organization A to develop an effort prediction model for maintenance change requests. Quite a few of the variables that were found to be relevant are known when the change request is being analysed. Still, further research is needed to determine formulas with enough predictive validity.



**Figure 4. Comparing the factors**

The dataset of organization B proved not very useful in explaining effort spent on maintenance change requests. In a sense, this is surprising, because the environments and the measurement programs are quite similar in many respects:

- The kind of applications maintained, the programming

languages and the supporting technology in both organizations are comparable.

- The size of the changes and the effort spent on changes is of the same order of magnitude.
- The measures taken in both organizations are comparable.
- The resulting factors from both datasets, though not exactly equal, look very similar, see figure 4. In both cases, the size of a change and the size of the components affected by a change, dominate. This concurs with other findings, e.g. [3, 8].
- Both measurement programs were set up and implemented in the same way. The success factors for measurement programs, as exemplified by Hall and Fenton [5], were kept in mind.

On the other hand, there are some differences between the measurement programs of organizations A and B, that might explain the differences in explanatory power of the datasets:

- As mentioned in section 4.2, we had no control over the accuracy of the effort data in organization B.
- As described in section 3.1, the maintenance process in organization B is quite dependent on how programmers and intermediaries divide analysis and design tasks between them. It is very well possible that this lack of standardization has a large impact on the usefulness of the data gathered.

There is no a priori evidence to suggest that the effort data for organization B is less reliable than those of organization A. This leaves us with the heterogeneity of the work processes in organization B as an explanation for the difference in explanatory power of the data. Hence, we conjecture that the consistent use of a standardized process is an important prerequisite for a successful measurement program.

Standardization does not necessarily mean that the process looks the same for all teams and all situations. It does mean that we know which variant of the process is being executed when. An example of this is the maintenance process in organization A, where one of the three teams uses a different variant of the process than the other two.

Finally, for organization A we found a rather strong relation between the type of attributes measured and the kind of activity in which these attributes play an important role: we measured quite some source code attributes, and the explained variance is especially noteworthy for the coding phase. This suggests that in order to improve overall prediction, we should look for additional variables that specifically bear upon analysis and testing.

## Acknowledgements

This research was partly supported by the Dutch Ministry of Economic Affairs, projects ‘Concrete Kit’, nr. ITU94045, and ‘KWINTES’, nr. ITU96024. Partners in this project are Cap Gemini, Twijnstra Gudde, the Tax and Customs Computer and Software Centre of the Dutch Tax and Customs Administration, and the Technical Universities of Delft and Eindhoven. We would like to thank the MSc students Kit Lam and Jack Hulscher who implemented the measurement programs.

## References

- [1] A. Abran and H. Nguyenkim. Measurement of the Maintenance Process from a Demand-based Perspective. *Software Maintenance: Research and Practice*, 5:64–90, 1993.
- [2] V. Basili, L. Briand, S. Condon, Y.-M. Kim, W. L. Melo, and J. D. Valett. Understanding and Predicting the Process of Software Maintenance Releases. In *Proceedings of the 18th International Conference on Software Engineering*, pages 464–474, Berlin, Germany, May 25-29, 1996. IEEE Computer Society Press.
- [3] W. M. Evanco. Modeling the Effort to Correct Faults. *The Journal of Systems and Software*, 29(1):75–84, Apr. 1995.
- [4] D. Gefen and S. L. Schneberger. The Non-Homogeneous Maintenance Periods: A Case Study of Software Modifications. In *Proceedings of the International Conference on Software Maintenance*, pages 134–141, Monterey, California, November 4-8, 1996. IEEE Computer Society.
- [5] T. Hall and N. Fenton. Implementing Effective Software Metrics Programs. *IEEE Software*, 14(2):55–65, March/April 1997.
- [6] J. Henry, R. Blasewitz, and D. Kettinger. Defining and Implementing a Measurement-based Software Maintenance Process. *Software Maintenance: Research and Practice*, 8(2):79–100, 1996.
- [7] M. Jørgensen. An Empirical Study of Software Maintenance Tasks. *Software Maintenance: Research and Practice*, 7:27–48, 1995.
- [8] F. Niessink and H. van Vliet. Predicting Maintenance Effort with Function Points. In M. J. Harrold and G. Visaggio, editors, *Proceedings of the International Conference on Software Maintenance*, pages 32–39, Bari, Italy, October 1-3, 1997. IEEE Computer Society.
- [9] F. Niessink and H. van Vliet. Towards Mature Measurement Programs. In P. Nesi and F. Lehner, editors, *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, pages 82–88, Florence, Italy, March 8-11, 1998. IEEE Computer Society.