**Technical Report**
CMU/SEI-96-TR-014
ESC-TR-96-014
June 1996

# A Controlled Experiment Measuring the Effect of Procedure Argument Type Checking on Programmer Productivity

Lutz Prechelt
Walter F. Tichy
Universität Karlsruhe, Germany

Disciplined Engineering Program



**Software Engineering Institute**
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/ENS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.



This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For informa tion on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides ac cess to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential con tractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

# Table of Contents

# A Controlled Experiment Measuring the Effect of Procedure Argument Type Checking on Programmer Productivity

**Abstract**: Type checking is considered an important mechanism for detecting programming errors, especially interface errors. This report describes an experiment to assess the error-detection capabilities of static intermodule type checking.

The experiment uses ANSI C and Kernighan&Ritchie (K&R) C. The relevant difference is that the ANSI C compiler checks module interfaces (i.e., the parameter lists of calls to external functions), whereas K&R C does not. The experiment employs a counterbalanced design in which each subject writes two non-trivial programs that interface with a complex library (Motif). Each subject writes one program in ANSI C and one in K&R C. The input to each compiler run is saved and manually analyzed for errors.

Results indicate that delivered ANSI C programs contain significantly fewer interface errors than delivered K&R C programs. Furthermore, after subjects have gained some familiarity with the interface they are using, ANSI C programmers remove errors faster and are more productive (measured in both time to completion and functionality implemented).

This report describes the design, setup, and results of the experiment including complete source code and error lists.

# 1. Introduction

Datatypes are an important concept in programming languages. A datatype is an interpretation applied to a datum, which otherwise would just be a string of bits. Datatypes are used to model the data space of a problem domain and are an important aid to programming and program understanding. A further benefit is type checking: A compiler or interpreter can determine whether a data item of a certain type is permissible in a given context, such as an expression or statement. If it is not, the compiler has detected an error in the program. It is the error-detection capability of type checking that is of interest in this paper.

There is some debate over whether dynamic type checking is preferable to static type checking, how strict the type checking should be, and whether explicitly declared types are more helpful than implicit ones. However, it seems that the benefits of type checking are virtually undisputed. Modern programming languages have evolved elaborate type systems and checking rules. In some languages, such as C, the type-checking rules were even strengthened in later versions. Furthermore, type theory is an active area of research.

However, it seems that the benefits of type checking are largely taken on faith or are based on personal anecdotes. For instance, Wirth states [Wirth 1994] that the type-checking facilities of Oberon had been most helpful in evolving the Oberon system. Many programmers can recall instances when type checking did or could have helped them. However, we could not find any reports in the literature on controlled, repeatable experiments that test whether type checking has any positive (or negative) effects. The cost and benefits of type checking are far from clear, because type checking is not free: It requires effort on behalf of the programmer in providing type information. Furthermore, there is some evidence that inspections might be more effective in finding errors than compilers [Humphrey 1995].

We conclude that the actual costs and benefits of type checking are largely unknown. This situation seems to be at odds with the importance assigned to the concept: Languages with type checking are widely used and the vast majority of practicing programmers are affected by the technique in their day-to-day work. The purpose of this paper is to provide initial, "hard" evidence about the effects of type checking. We describe a repeatable and controlled experiment that confirms some positive effects: First, when applied to interfaces, type checking reduces the number of errors remaining in delivered programs. Second, when programmers use a familiar interface, type checking helps them remove errors more quickly and increases their productivity.

Definitive knowledge about positive effects of type checking can be useful in two ways: First, we still lack a useful scientific model of the programming process. Such a model is a prerequisite for understanding the overall software production process. Understanding the types, frequencies, and circumstances of programmer errors is an important ingredient of such a model. Second, there are still many environments where type checking is missing or incomplete, and such knowledge will produce pressure to close these gaps. For instance it may pay off to invest in discriminating between the many kinds of integer values that occur in interfaces, such as cardinal numbers, indices, differences, etc.

# 2. Related Work

Work on error classification and detection obviously has a bearing on our experiment. Two publications describe and analyze the typical errors in programs written by novices [Ebrahimi 1994; Spohrer and Soloway 1986]. The results are not necessarily relevant for professional programmers. Furthermore, type errors do not play an important role in these studies.

Error-type analyses have also been performed in larger scale software development settings. Type checking has not been a concern in these studies, but in some cases related information can be derived. For instance, Basili and Perricone report that 39 percent of all errors in a 90.000 line FORTRAN project were interface errors [Basili and Perricone 1984]. We conjecture that some proportion of these could have been found by type checking.

The error-detection capabilities of testing methods is a question that has attracted considerable interest [Frankl and Weiss 1993]. The errors found by testing are those that already passed the type checks, so the results from these studies are hardly applicable here.

Several studies have compared the productivity effects of different programming languages. Most of these studies used programmers with little experience and very small programming tasks [Ebrahimi 1994]. Others used somewhat larger tasks and experienced programmers, but lacked proper experimental control [Hudak and Jones 1994]. All of these studies have the inherent problem that they are confounded by too many factors to draw conclusions regarding type checking.

We are aware of only one closely related experiment, the Snickering Type Checking Experiment[1] with the Mesa language. In that work, compiler-generated error messages involving types were diverted to a secret file. A programmer working with this compiler on two different programs was shown the error messages after he had finished the programs and was asked to estimate how much time he would have saved had he seen the messages right away. Interestingly, the programmer had independently removed all the errors detected by the type checker. He claimed that on one program, which was 100% his own work, type checking would not have helped appreciably. On another program which involved interfacing to a complicated library, he estimated that type checking would have saved 50% of total development time. It is obvious that this type of study has many flaws. But to our knowledge it was never repeated in a more controlled setting.

It appears that the cost and benefits of type checking have not been studied systematically.

[1] J.H. Morris, unpublished, 1978

# 3. Design of the Experiment

The idea behind the experiment is the following: Let experienced programmers solve programming problems involving a complex library. To control for the type-checking/no-type-checking variable, let every subject solve one problem with K&R C, and another with ANSI C. Save the inputs to all compiler runs for later error analysis.

A number of observations regarding the realism of the setup are in order. A simple task means that the difficulties observed will stem from using the library, not from solving the task itself. Thus, most errors will occur when interfacing to the library, where the effects of type checking are thought to be most pronounced. Furthermore, using a complex library is similar to the development of a module within a larger project, where many imported interfaces must be handled. To ensure that the results would not be confounded by problems with the language, we used experienced programmers familiar with the programming language. However, the programmers had no experience with the library — another similarity with realistic software development, in which new modules are often written within a relatively foreign context.

To balance for both learning effects and intersubject ability differences, we used a counter-balanced design: There were two independent problems to be solved (A and B, as described below) and two treatments (ANSI C and K&R C). Each subject had to solve both problems, each with a different language. Thus, there are two experimental groups: Group 1 solves A(ANSI)+B(KR) (in this order) and group 2 solves B(ANSI)+A(KR).

Controlling for the sequence of problems and languages creates another two groups; see Table 3-1. The dependent variables are described in Section 3.4.

Subjects were assigned to groups in round-robin fashion.

## 3.1. Tasks

**Problem A (**$2 \times 2$ **Matrixinversion):** Open a window with four text fields arranged in a $2 \times 2$ pattern plus an “Invert” and a “Quit” button. See Figure 3-1. “Quit” exits the program and closes the window. The text fields represent a matrix of real values. The values can be entered and edited. When the “Invert” button is pressed, replace the values by the coefficients of the corresponding inverted matrix, or print an error message if the matrix is not invertible. The formula for $2 \times 2$ matrix inversion was given.

| | first problem A<br>second problem B | first problem B<br>second problem A |
|---|---|---|
| first ANSI C<br>then K&R C | Group 1 | Group 2 |
| first K&R C<br>then ANSI C | Group 3 | Group 4 |

**Table 3-1 Tasks and compilers assigned to the four groups of subjects**

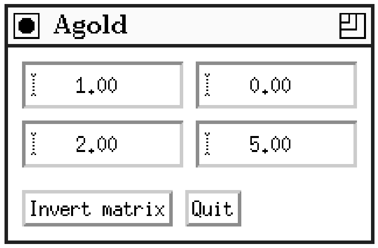


**Figure 3-1 Problem A (2 × 2 matrix inversion)**

**Problem B (File Browser):** Open a window with a menubar containing a single menu. The menu entry “Select file” opens a file-selector box. The entry “Open selected file” pops up a separate, scrollable window and displays the contents of the file previously selected in the file selector box. “Quit” exits the program and closes all its windows. See Figure 3-2.

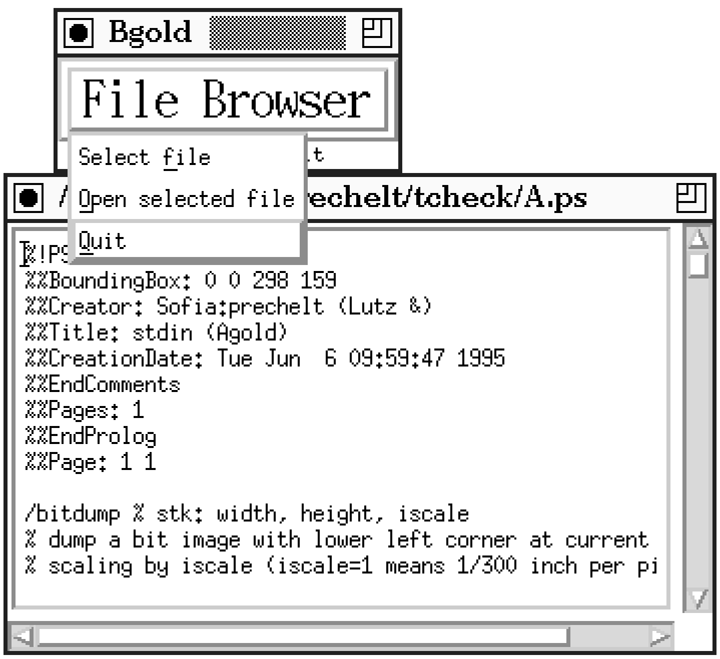


**Figure 3-2 Problem B (File browser)**

For solving the tasks, the subjects did not use native Motif, but a special wrapper library; see Appendix D. This library has operations similar to those of Motif, but with improved type checking. For instance the library has no functions with variable-length arguments lists, which Motif uses often. It also imposes types on the resource-name constants that reflect the type of the resource; in Motif, all resources are handled typelessly. There was also some simplification through additional convenience functions. For instance, there was a function for creating a RowColumnManager and setting its orientation and packing mode in one call.

The tasks, although quite small, were not at all trivial. The subjects had to understand several important concepts of Motif programming (such as widget, resource, and callback function).

They had to learn to use them from abstract documentation only, without example programs (as is typically the case in practice).

## 3.2. Subjects

40 unpaid volunteers participated in the study. Of these, 6 were removed from the sample: One deleted his protocol files, one was obviously inexperienced (took almost 10 times as long as the others), and 4 worked on only one of the two problems. After this mortality, the A/B groups had 8+8 subjects, the B/A groups had 11+7 subjects. We consider this to be still sufficiently balanced.

The remaining 34 subjects had the following education. Of these 2 were postdocs in computer science (CS); 19 were PhD students in CS and had completed an MS degree in CS; another subject was also a CS PhD student but held an MS in physics; 12 subjects were CS graduate students with a BS in CS.

The 34 subjects had between 4 and 19 years of programming experience ($\mu = 10.0$) and all but 11 of them had written at least 3000 lines in C (all but one at least 300 lines). Only 8 of the subjects had any programming experience with X-Windows or Motif; only 3 of them had written more than 300 lines in X-Windows or Motif. For detailed information about the subjects see Appendix E.

## 3.3. Setup

Each subject received two written documents and one instruction sheet and was then left alone at a Sun-4 workstation to solve the two problems. The subjects were told to use roughly one hour per problem, but no time limit was enforced. Subjects could stop working even if the programs were not operational.

The instruction sheet was a one-page description of the global steps involved in the experiment: “Read sections 1 to 3 of the instruction document; fill in the questionnaire in section 2; initialize your working environment by typing `make TC1`; solve problem A by. . . ” and so on. The subjects obtained the following materials, most of them both on paper and in files:

1. a half-page introduction to the purpose of the experiment
2. a questionnaire about the background of the subject
3. specifications of the two tasks plus the program skeleton for them
4. an introduction to Motif programming (one page) and some useful commands (for example how to to search manuals online)
5. a manual listing first the names of all types, constants, and functions that might be required, followed by descriptions of each of them including the signature, semantic description, and several kinds of cross-references. The document also included introductions to the basic concepts of Motif and X-Windows. This manual was hand tailored to contain all information required to solve the tasks and hardly anything else.
6. a questionnaire about the experiment (to be filled in at the end)

Subjects could also execute a “gold” program for each task. A gold program solved its task completely and correctly and could be used as a backup for the verbal specifications. Subjects were told to write programs that duplicated the behavior of the gold programs. The source code of the gold programs is shown in Appendices A.1. and B.1.

The subjects did not have to write the programs from scratch. Instead, they were given a program skeleton that contained all necessary `#include` commands, variable and function declarations, and some initialization statements. In addition, the skeleton contained pseudocode describing step by step what statements had to be inserted to complete the program. The subjects’ task was to find out which functions they had to use and which arguments to supply. Almost all statements were function calls.

The following is an example of a pseudostatment in the skeleton.

/* Register callback-function ’button_pushed’ for the ’invert’ button with the number 1 as ’client_data’ */

It can be implemented thus:

XtAddCallbackF(invert, XmCactivateCallback, button_pushed, (XtPointer)1);

There were only few variations possible in the implementation of the pseudocode. The complete program skeletons are shown in the Appendices A.2. and B.2.

The programming environment captured all program versions submitted for compilation along with a timestamp and the messages produced by the compiler and linker. A timestamp for the start and the end of the work phase for each problem was also written to the protocol file.

The environment was set up to call the standard C compiler of SunOS 4.1.3 using the command `cc -c -g` for the K&R tasks and version 2.7.0 of the GNU C compiler using `gcc -c -g -ansi -pedantic -W -Wimplicit -Wreturn-type` for the ANSI C tasks.

## 3.4. Observed Variables

After the experiment was finished, each program version in the protocol files was annotated by hand. Each different programming error that occurred in the programs was identified and given a unique number. For instance, for the call to `XtAddCallbackF` shown above, there were 15 different error numbers, including 4 for wrong argument types, 4 for wrong argument objects with correct type, and another 7 for more specialized errors. The complete lists and descriptions of the error numbers for both problems are shown in Appendices A.1. and B.1.

Each program version was annotated with the errors introduced, removed, or changed (without correcting them).

Additional annotations counted the number of type errors, other semantic errors, and syntactic errors that actually provoked one or more error messages from the compiler or linker. The timestamps were corrected for pauses that lasted more than 10 minutes. Summary statistics were computed, for which each error was classified into one of the following categories:

*non-error*: False negatives–variations that the annotators considered errors, but were in fact correct. This class will be ignored in the following.

*comp*: Errors that had to be removed before the program would pass the compiler and linker, even for K&R C. This class will be ignored.

*slight*: Errors resulting in slightly wrong functionality of the program, but so minor that the programmers probably felt no need to correct them. This class will be ignored.

*invis*: Errors that are *invisible*, i.e., they do not compromise functionality, but only because of unspecified properties of the library implementation. Changes in the library implementation may result in a misbehaving program. Example: Supplying the integer constant `PACK_COLUMN` instead of the expected Boolean value `True` works correctly, because (and as long as) the constant happens to have a non-zero value. This class of errors will be ignored.

*invisD*: same as *invis*, except that the errors will be detected by ANSI C parameter type checking (but not by K&R C).

*severe*: Errors resulting in significant deviations from the prescribed functionality.

*severeD*: same as *severe*, except that the errors will be detected by ANSI C parameter type checking (but not by K&R C).

These categories are mutually exclusive. Unless otherwise noted, the error statistics discussed below are computed based on the sum of *severe*, *severeD*, and *invisD*. The assignment of errors to the above categories is shown in Appendix C.

Other metrics observed were the number of compilation cycles (versions) and time to completion, i.e., the time taken by the subjects before delivering the program (whether complete and correct or not).

From these metrics and annotations, additional statistics were computed. For instance the frequency of error insertion and removal, the number of attempts made before an error was finally removed, the time an error remained in the program ("lifetime"), and the number and type of errors remaining in the final program version.

For measuring productivity and unimplemented functionality, we define a *functionality unit (FU)* to be a single statement in the gold programs. Using the gold programs as a reference normalizes the cases in which subjects used more than one statement instead. FUs are thus a better measure of program volume than lines of code. Gold program A has 16 FUs, B has 11.

We also annotated the programs with the number of *gaps*, i.e., the number of missing FUs. An FU is counted as missing if a subject made no attempt to implement it. From this, it is easy to derive the number of FUs implemented in a program.

## 3.5. Internal and External Validity

The following problems might threaten the internal validity of the experiment (the correctness of the observations):

1. Error messages produced by the two compilers might differ for the same error, and this might influence productivity. Our subjective judgement here is that the error messages of both compilers are comparable in quality, at least for the purposes of this experiment.
2. There may be annotation errors. To insure consistency, all annotations were made by the same person. The annotations were cross-checked first with a simple consistency checker, and then some of them were checked manually. The number of annotation errors found in the manual check was negligible (4%).

The following problems might limit external validity of the experiment, i.e., the generalizability of our results:

1. The subjects were not professional software engineers. However, they were quite experienced programmers and held degrees (many of them advanced) in computer science.
2. The results may be domain dependent. This objection cannot be ruled out. This experiment should therefore be repeated in domains other than graphical user interfaces.
3. The results may not apply to situations in which the subjects are very familiar with the interfaces used.

Despite these problems, we expect that the scenario chosen in the experiment is nevertheless similar to many real situations with respect to type-checking errors.

# 4. Results and Discussion

Most of the statistics of interest in this study have clearly non-normal distributions and sometimes severe outliers. Therefore, we present medians (to be precise: an interpolated 50% quantile) rather than arithmetic means. Where most of the median values are zero, higher quantiles are given.

The results are shown in Table 4-1. There are 13 different statistics in 3 main columns. The first column shows the statistics for both tasks, independent of order. The second and third columns reflect the observations for those tasks that were tackled first and second, respectively. These columns can be used to assess the learning effect. Each main column reports the medians (or higher quantiles where indicated) for the tasks programmed with ANSI C and K&R C plus the $p$-value. The $p$-value is the result of the Wilcoxon Rank Sum Test[1] and can be interpreted as the probability that the observed differences occurred by chance. A difference in the median is considered significant if $p \leq 0.05$. Significant results are marked in

[1]This test, also known as Mann-Whitney U Test, was chosen because the distributions of the variables are more or less logistic, rather than, say, normal or double exponential.

| | Statistic | both tasks ANSI | both tasks K&R | both tasks $p =$ | 1st task ANSI | 1st task K&R | 1st task $p =$ | 2nd task ANSI | 2nd task K&R | 2nd task $p =$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | hours to completion | 1.3 | 1.35 | 0.49 | 1.6 | 1.6 | 0.83 | 0.9 | 1.3 | **0.018** |
| 2 | #versions | 15 | 16 | 0.84 | 19 | 21 | 0.63 | 12.5 | 13 | 0.16 |
| 3 | #type error messages/hour | 6.3 | 1.1 | **0.0000** | 4.3 | 1.2 | **0.0007** | 7.7 | 1.0 | **0.0006** |
| 4 | #error insertions/hour | 5.6 | 6.5 | 0.35 | 4.0 | 4.2 | 0.28 | 6.3 | 6.8 | 0.75 |
| 5 | #error removals/hour | 4.15 | 3.95 | 0.69 | 4.0 | 4.2 | 0.97 | 4.9 | 3.7 | 0.60 |
| 6 | sum of accumulated error lifetime | 1.6 | 2.55 | **0.035** | 2.2 | 3.6 | 0.26 | 0.8 | 2.2 | **0.025** |
| 7 | #right, then wrong again (75% quant.) | 1.0 | 1.0 | 0.12 | 1.0 | 1.0 | 0.82 | 0.0 | 1.0 | **0.009** |
| 8 | #remaining errs in delivered program | 1.0 | 2.0 | **0.016** | 1.0 | 2.0 | 0.32 | 1.0 | 2.0 | **0.031** |
| 9 | — for *invisD* only (90% quantile) | 0.0 | 1.0 | **0.04** | 0.0 | 1.4 | **0.048** | 0.0 | 0.0 | 0.41 |
| 10 | — for *severe* only | 1.0 | 1.0 | 0.66 | 1.0 | 0.0 | 0.74 | 1.0 | 1.0 | 0.65 |
| 11 | — for *severeD* only | 0.0 | 1.0 | **0.0001** | 0.0 | 1.0 | **0.015** | 0.0 | 1.0 | **0.0022** |
| 12 | #gaps (75% quantile) | 0.25 | 0.0 | 0.35 | 1.5 | 0.0 | 0.26 | 0.0 | 0.0 | 0.70 |
| 13 | FU/h | 8.6 | 9.7 | 0.93 | 7.21 | 8.5 | 0.31 | 12.8 | 10.7 | 0.061 |

**Table 4-1 Medians (or other quantiles as indicated) of statistics for ANSI C vs. K&R C versions of programs and $p$-values for statistical significance of Wilcoxon Rank Sum Tests of the two. Values under 0.05 indicate significant differences of the medians. Column pairs are for 1st+2nd, 1st, and 2nd problem tackled chronologically by each subject, respectively. All entries include data points for both problem A and problem B.**

boldface in the table. When the $p$-value is not significant, nothing can be said; in other words there may or may not be a difference.

The first statistic, time to completion, shows that there is no significant difference between ANSI C and K&R C for the first task and both tasks together. The combined time spent for the second task is shorter than for the first ($p = 0.0012$, not shown in the table), indicating a learning effect. In the second task, ANSI C is significantly more productive. A plausible explanation is as that when they started, programmers did not have a good understanding of the library and were struggling more with the concepts than with the interface itself. A lack of understanding is also borne out by the protocols. Type checking is unlikely to help gain a better understanding. Type checks became useful only after programmers had overcome the initial learning hurdle.

Statistic 2, the number of program versions compiled, does not show a significant difference. Entry 3 shows that the ANSI C compiler does indeed flag type errors significantly more often than the K&R compiler does. Each type error was counted only once per compilation for this statistic, no matter whether it produced one or several messages. Messages produced for other semantic or for syntactic errors were ignored.

Entries 4 to 7 are statistics that describe the internal error processes, all based on the sum of the error categories *invisD*, *severe*, and *severeD*. The frequency of error insertion and removal (entries 4 and 5) show no significant differences. The other two show some advantage for ANSI C, and it is again most pronounced in task 2, confirming the learning effect.

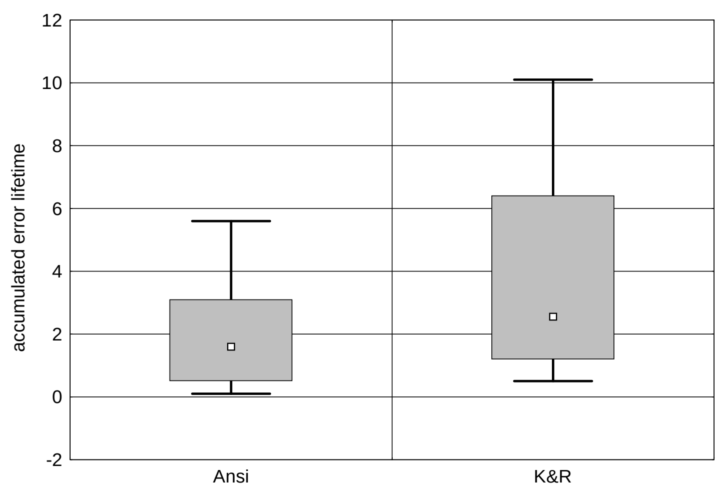


**Figure 4-1 Boxplots of accumulated error lifetime (in hours) over both tasks for ANSI C (left boxplot) and K&R C (right boxplot). The upper and lower whiskers mark the 95% and 5% quantiles, the upper and lower edges of the box mark the 75% and 25% quantiles, and the dot marks the 50% quantile (median). All other boxplots following below have the same structure.**

The total lifetime of all errors during programming (entry 6) is shorter for ANSI C overall and in the 2nd task. The distributions of accumulated lifetime over both tasks are also shown as boxplots in Figure 4-1. As we see, the K&R total error lifetimes are usually higher and spread over a much wider range.

The number of errors introduced in previously correct or repaired parts of a program (entry 7) is significantly higher for K&R C in the 2nd task.

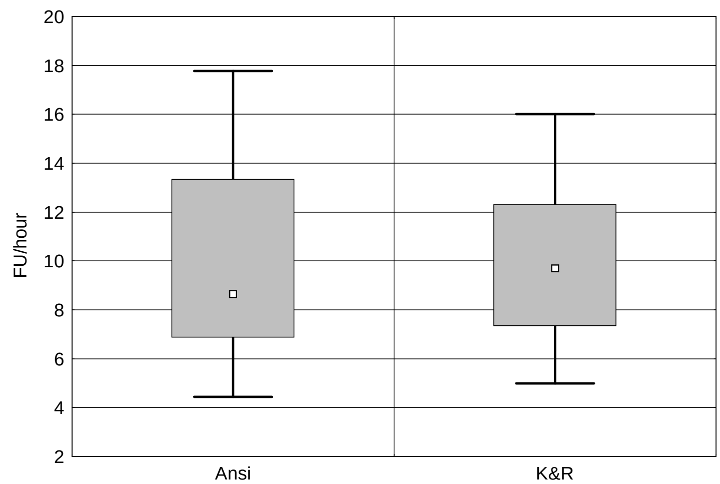


**Figure 4-2 Boxplots of productivity (in FU/hour) over both tasks.**

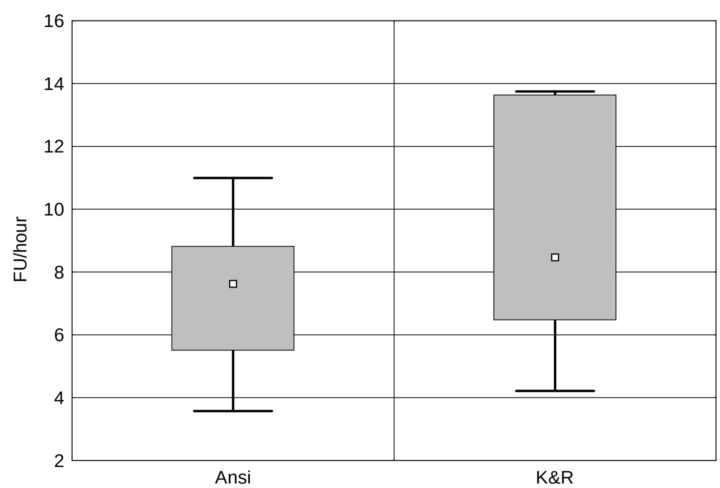


**Figure 4-3 Boxplots of productivity (in FU/hour) for first task.**

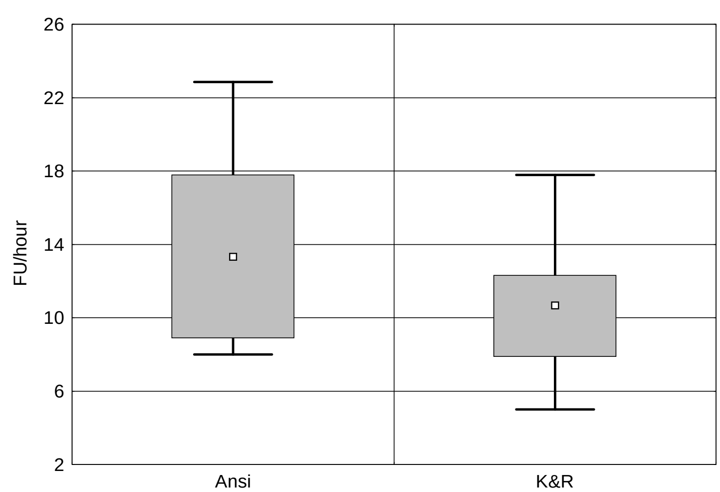


**Figure 4-4 Boxplots of productivity (in FU/hour) for second task.**

There are no significant differences in the number of gaps in the delivered programs (entry 12). However, the $p$-value of 0.061 for productivity (entry 13) in the second task strongly suggests that ANSI C is helpful for programmers after the initial interface learning phase. The combined (both languages) productivity rises significantly from the first task to the second task ($p = 0.0001$, not shown in the table); this was also reported by the subjects.

The distributions of productivity measured in FU/hour are shown in Figures 4-2 to 4-4. We see that ANSI C makes for a more pronounced increase in productivity from the first task to the second than does K&R C.

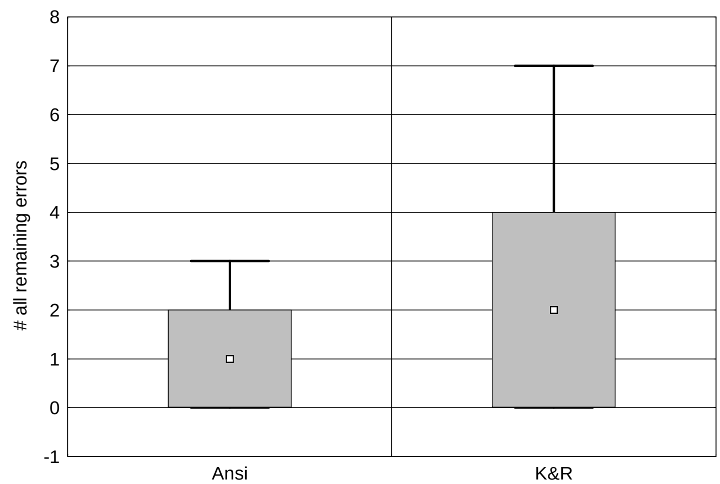


**Figure 4-5 Boxplots of total number of remaining errors in delivered programs over both tasks.**

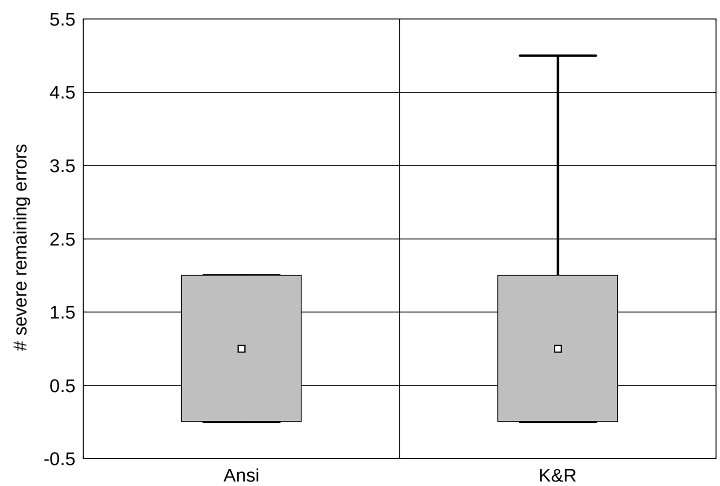


**Figure 4-6 Boxplots of number of remaining *severe* errors in delivered programs over both tasks.**

A clear advantage of ANSI C over K&R C is the number of errors still present in the delivered program (entry 8). As entries 9 to 11 indicate, this result stems from the direct detection of errors through type checking; little or no reduction of non-detectable errors (entry 10) is achieved. The both-task distributions for entries 8, 10, and 11 are shown in Figures 4-5 to 4-7. As we see there, the distributions for *severe* errors differ only in the upper tail (Figure 4-6), whereas the distributions for the *severeD* errors differ dramatically in favor of ANSI C (Figure 4-7), resulting in a significant overall advantage for ANSI C (Figure 4-5).

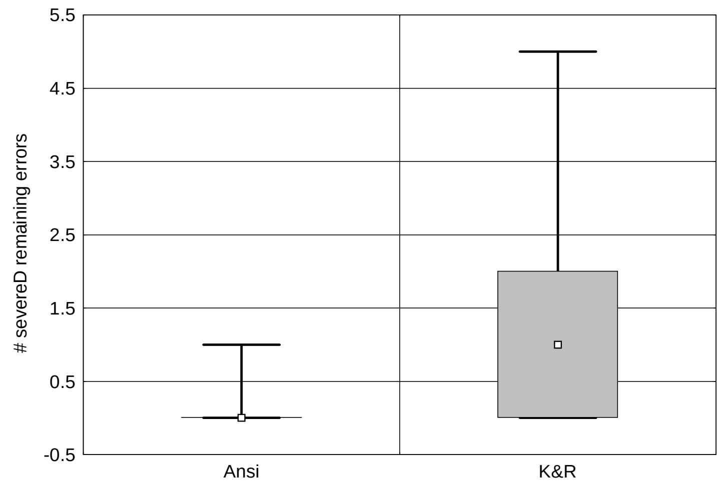


**Figure 4-7 Boxplots of number of remaining *severeD* errors in delivered programs over both tasks.**

A detailed analysis of the errors remaining in the delivered programs is shown in Table 4-2 on page 15. Only errors that remained in more than one delivered program are included in the table. There are no apparent patterns in this data except for those that are already

| error | det? | KR | ANSI |
|---|---|---|---|
| O101 | no | 2 | 0 |
| O142 | no | 2 | 1 |
| O155 | no | 2 | 0 |
| O165 | no | 2 | 0 |
| O604 | no | 0 | 2 |
| $sum(O) =$ | | 8 | 3 |
| P143 | no | 3 | 3 |
| P234 | ANSI | 2 | 0 |
| P271 | ANSI | 3 | 0 |
| P284 | no | 1 | 1 |
| P678 | ANSI | 2 | 2 |
| $sum(P) =$ | | 11 | 6 |

| error | det? | KR | ANSI |
|---|---|---|---|
| T152 | ANSI | 2 | 0 |
| T162 | ANSI | 2 | 0 |
| T172 | ANSI | 1 | 1 |
| T182 | ANSI | 1 | 1 |
| T231 | ANSI | 7 | 0 |
| T236 | ANSI | 2 | 0 |
| T273 | no | 0 | 2 |
| T277 | ANSI | 2 | 0 |
| T524 | ANSI | 3 | 1 |
| T541 | ANSI | 2 | 0 |
| T585 | ANSI | 2 | 0 |
| T679 | ANSI | 2 | 2 |
| $sum(T) =$ | | 26 | 7 |

| error | det? | KR | ANSI |
|---|---|---|---|
| V117 | no | 3 | 0 |
| V126 | no | 1 | 4 |
| V127 | no | 5 | 1 |
| V128 | no | 1 | 1 |
| V136 | no | 2 | 1 |
| V137 | no | 5 | 1 |
| V146 | no | 5 | 5 |
| V716 | no | 4 | 7 |
| $sum(V) =$ | | 26 | 20 |

**Table 4-2 Number of occurrences of *severe* and *severeD* errors that remained in more than one *delivered* program. "error": error type and number, see Appendices A.1., B.1., and C. "det?": whether error was compiler-detectable. "KR," "ANSI": number of delivered K&R or ANSI programs with this error.**

apparent from the cumulative data discussed above (P and T errors). For O and V errors, the differences between K&R and ANSI in the location of the frequency distributions are not statistically significant (Wilcoxon Matched Pairs Test, $p \approx 0.28$ for O errors, $p \approx 0.35$ for V errors).

There were no significant differences between the two tasks. All of the above results hardly change if one considers the tasks A and B separately (not shown).

Finally, the subjective impressions of the subjects as reported in the questionnaires are as follows: 26 of the subjects (79%) noted a learning effect from the first program to the second. 9 subjects (27%) reported that they found the ANSI C type checking very helpful, 11 (33%) found it considerably helpful, 4 (12%) found it almost not helpful, 5 (15%) found it not at all helpful. 4 subjects could not decide and 1 questionnaire was lost.

# 5. Conclusions and Further Work

Our experiment confirms the following hypotheses:

1. Type checking can reduce the number of interface errors in delivered programs.
2. When an interface is used, type checking can increase productivity, provided the programmer has gained a basic understanding of the interface.

One must be careful generalizing the results of this study to other situations. For instance, the experiment is unsuitable for determining the proportion of interface errors in an overall mix of errors, because it was designed to prevent errors other than interface errors. Hence it is unclear how large the differences will be if error classes such as declaration errors, initialization errors, algorithmic errors, or control-flow errors are included.

Nevertheless, the experiment suggests that for many realistic programming tasks, type checking of interfaces improves both productivity and program quality. Furthermore, some of the resources otherwise expended on inspecting interfaces might be allocated to other tasks.

Further work should repeat similar error analyses in different settings (e.g. tasks with complex data flow or object-oriented languages). In particular, it would be interesting to compare productivity and error rates under compile-time type checking, runtime type checking, and type inference. Other important questions concern the influence of a disciplined programming process such as the Personal Software Process$^{SM}$ [Humphrey 1995][1]. Finally, an analysis of the errors occurring in practice might help devise more effective error detection mechanisms.


## Acknowledgements

Thanks to the experimental subjects. Many thanks in particular to Paul Lukowicz for patiently guinea-pigging the experimental setup and to SAS Institute, makers of JMP, for creating a neat statistical tool.


[1]Personal Software Process is a service mark of Carnegie Mellon University

# Appendix A. Problem A

## A.1. Error Numbers and Gold Program

The following list shows the body of the gold program for problem A. Interspersed between the statements are the error numbers and error descriptions, always relating to the *preceding* statement or statements. The number given after each error number is the error incidence, i.e., the total number of error insertions for this error number found over all protocols. For the errors that have different error codes for the two compilers, both numbers (detectable instances, undetectable instances) are shown separately.

```
int main (argc, argv)
  int argc;
  char *argv[];
{
  Widget        toplevel, manager, square, buttons, invert, quit;
  XtAppContext  app;
  XmString  invertlabel, quitlabel;

  /*---------- 1. initialize X and Motif --------------------------------*/
  /* (already complete, should not be changed) */
  toplevel = XtVaAppInitialize (&app, "Hello", NULL, 0,
        &argc, argv, fallbacks, NULL);

  /*---------- 2. create and configure widgets --------------------------*/
```

*O101* 7 local 'mw' declared (hides global 'mw', which is used in button_pushed)
*O102* 1 local widgets for TextFields declared, but not with name 'mw'

```
  manager = XmCreateRowColumnManagerOCP ("manager", toplevel, XmVERTICAL,
                                        1, False);
```

*MY111* 1 numColumns missing
*Z112* 1 XmPACK_COLUMN or XmPACK_TIGHT used instead of Bool
*P113* 1 XtSetValue called before Create
*TZ114* 0/1 first arg not String
*O115* 0 not 'toplevel'
*V116* 0 XmHORIZONTAL
*V117* 5 True
*TZ118* 1/0 no integer for numColumns [the value does not matter here!]
*TZ119* 0/1 XtSetxxValue called with wrong xx
*O301* 2 result not assigned (or to wrong variable)
*P302* 0 nonsense procedure called to set resource
*E303* 0 direct assignment to resource attempted
*TZ304* 0/0 resource names as strings, e.g. "XmCorientation" used with XtSetxxValue
*P305* 0/0 XmCreateRowColumnManager (not OCP) called, but with all 5 args

```
  square = XmCreateRowColumnManagerOCP ("square", manager, XmHORIZONTAL,
                                       2, True);
```

*MY121* 0 numColumns missing
*Z122* 5 XmPACK_COLUMN or XmPACK_TIGHT used instead of Bool
*P123* 1 XtSetValue called before Create
*TZ124* 0/1 first arg not String
*O125* 2 not 'manager'

| | | |
|---|---|---|
| *V126* | 12 | XmVERTICAL or no orientation given |
| *V127* | 12 | False |
| *V128* | 8 | not 2 columns |
| *TZ129* | 1/1 | XtSetxxValue called with wrong xx |
| *O311* | 0 | result not assigned (or to wrong variable) |
| *P312* | 2 | nonsense procedure called to set resource |
| *E313* | 1 | direct assignment to resource attempted |
| *TZ314* | 0/1 | resource names as strings, e.g. "XmCorientation" used with XtSetxxValue |
| *P315* | 0 | XmCreateRowColumnManager (not OCP) called, but with all 5 args |

```
buttons = XmCreateRowColumnManagerOCP ("buttons", manager, XmHORIZONTAL,
                                        1, False);
```

| | | |
|---|---|---|
| *MY131* | 0/0 | numColumns missing |
| *Z132* | 0/1 | XmPACK_COLUMN or XmPACK_TIGHT used instead of Bool |
| *P133* | 1 | XtSetValue called before Create |
| *TZ134* | 0/0 | first arg not String |
| *O135* | 1 | not 'manager' |
| *V136* | 10 | XmVERTICAL or no orientation given |
| *V137* | 9 | True |
| *TZ139* | 2/2 | XtSetxxValue called with wrong xx |
| *O321* | 1 | result not assigned (or to wrong variable) |
| *P322* | 1 | nonsense procedure called to set resource |
| *E323* | 0 | direct assignment to resource attempted |
| *TZ324* | 1 | resource names as strings, e.g. "XmCorientation" used with XtSetxxValue |
| *P325* | 1 | XmCreateRowColumnManager (not OCP) called, but with all 5 args |

```
mw[0] = XmCreateTextFieldWidgetW ("aw", square, 100, "a");
mw[1] = XmCreateTextFieldWidgetW ("bw", square, 100, "b");
mw[2] = XmCreateTextFieldWidgetW ("cw", square, 100, "c");
mw[3] = XmCreateTextFieldWidgetW ("dw", square, 100, "d");
```

| | | |
|---|---|---|
| *TZ141* | 1/1 | 4th arg not String |
| *O142* | 13 | not 'square' |
| *P143* | 8 | XmCreateTextFieldWidget used (no 'W', cannot set width later!) |
| *TZ144* | 0/0 | first arg not String |
| *O145* | 1 | missing assignment |
| *V146* | 13 | Initialization not "a", "b", "c", "d" |
| *O147* | 2 | result assigned to mw (without subscription) |
| *MY148* | 1/0 | 3rd arg missing |
| *TZ149* | 0/0 | XtSetxxValue called with wrong xx |
| *O331* | 2 | result not assigned at all |
| *P332* | 1 | nonsense procedure called to set resource |
| *E333* | 1 | direct assignment to resource attempted |
| *TZ334* | 0 | resource names as strings, e.g. "XmCorientation" used with XtSetxxValue |
| *P335* | 1 | magic effect expected from weird argument to first param |
| *D336* | 2 | wrong procedure name, e.g. XmCreateTextFieldW |

```
invert = XmCreatePushButtonL ("invert", buttons,
                              XmStringCreateLocalized ("Invert matrix"));
```

| | | |
|---|---|---|
| *D151* | 4 | XmCreatePushButton (without 'L') |
| *TZ152* | 7/12 | normal String instead of XmString |
| *TZ153* | 0/2 | XmStringCreate called with non-String first or second arg |
| *TZ154* | 1/2 | first arg not String |
| *O155* | 6 | not 'buttons' |
| *G156* | 3 | uninitialized 'invertlabel' used |
| *P157* | 1 | nonexisting function XmString called (or cast to) to create third arg |
| *MY158* | 1/0 | no second arg to XmStringCreate |

*O159* 1 result not assigned or assigned to wrong object
*C351* 0 sets XmCwidth explicitly
*D352* 0 nonexisting function called, e.g. XmPushButton
*TZ353* 1/0 second arg not Widget

```
  quit = XmCreatePushButtonL ("quit", buttons,
                              XmStringCreateLocalized ("Quit"));
```

*D161* 5 XmCreatePushButton (without 'L')
*TZ162* 7/12 normal String instead of XmString
*TZ163* 0/1 XmStringCreate called with non-String first or second arg
*TZ164* 1/2 first arg not String
*O165* 6 not 'buttons'
*G166* 3 uninitialized 'quitlabel' used
*P167* 1 nonexisting function XmString called (or cast to) to create third arg
*MY168* 1/0 no second arg to XmStringCreate
*O169* 1 result not assigned or assigned to wrong object
*C361* 1 sets XmCwidth explicitly
*D362* 0 nonexisting function called, e.g. XmPushButton
*TZ363* 1/0 second arg not Widget

```
  /*---------- 3. register callback functions --------------------------*/

  XtAddCallbackF (invert, XmCactivateCallback, button_pushed, (XtPointer)1);
```

*D171* 2 XtPointer misspelled
*TZ172* 5/8 no XmFuncResourcename
*TZ173* 1/1 no Callback function pointer
*TZ174* 6/7 no XtPointer
*O175* 2 not 'invert' [see 196]
*V176* 6 not 'XmCactivateCallback'
*V177* 0 not button_pushed (other name used)
*O178* 3 &d instead of 1 given, with local or non-constant d
*TZ179* 1/1 4th: no pointer or int
*T191* 2 tried to supply prototype for button_pushed
*T192* 5 tried to supply arguments or () for button_pushed
*D193* 1 used undeclared object for arg 4
*E194* 1 nonsense function called for arg 2
*E195* 1 cast to type client_data or similar attempted
*TZ196* 1 first arg not a widget

```
  XtAddCallbackF (quit, XmCactivateCallback, button_pushed, (XtPointer)99);
```

*D181* 2 XtPointer misspelled
*TZ182* 3/6 no XmFuncResourcename
*TZ183* 0/1 no Callback function pointer
*TZ184* 4/6 no XtPointer
*O185* 5 not 'quit' [see 206]
*V186* 3 not 'XmCactivateCallback'
*V187* 0 not 'button_pushed' (other name used)
*O188* 3 &d instead of 1 given, with local or non-constant d
*TZ189* 0/1 4th: no pointer or int
*TZ201* 2/0 tried to supply prototype for button_pushed
*TZ202* 2/0 tried to supply arguments or () for button_pushed
*D203* 1 used undeclared object for arg 4
*E204* 0 nonsense function called for arg 2
*E205* 1 cast to type client_data or similar attempted
*TZ206* 1/0 first arg not a widget

```
  /*---------- 4. realize widgets and turn control to X event loop ------*/
```

```
  /* (already complete, should not be changed) */
  XtRealizeWidget (toplevel);
  XtAppMainLoop (app);
  return (0);
}

/************************ Functions ************************/

void button_pushed (Widget widget, XtPointer client_data, XtPointer call_data)
{
  double mat[4], new[4], det;
  String s;
  if ((int)client_data == 99) {
```

*V211* 6 not 99 or whatever is submitted above
*T212* 12 missing cast
*E213* 5 client_data wrongly dereferenced (or wrongly not dereferenced)
*C214* 2 recursive call to 'button_pushed' or call to 'cbf'
*E215* 1 = instead of ==
*O216* 2 call_data used instead of client_data

```
     exit (0);
```

*E341* 4 exit; (no argument and no parentheses) [exit() is acceptable]

```
  }
  else if ((int)client_data == 1) {
```

*V221* 3 not 1 or whatever is submitted above
*T222* 10 missing cast
*E223* 5 client_dataf wrongly [not] dereferenced
*E225* 1 = instead of ==
*O226* 0 call_data used instead of client_data

```
    int i;
    for (i = 0; i <= 3; i++) {
      XtGetStringValue (mw[i], XmCvalue, &s);
```

*TZ231* 12/8 value instead of pointer used, e.g. s instead of &s
*TZ232* 3/1 dereferenced (or a Typename) instead of referenced, e.g. *s instead of &s
*E233* 1 tried to assign result of XtGetValue to something
*P234* 14 XtGetFloatValue called
*P235* 2 XtGetIntValue called
*TZ236* 0/5 no XmStringResourceName
*O237* 0 not 'XmCvalue'
*TZ238* 1/0 &mat[i] or other float used with XtGetStringValue
*TZ239* 0/0 first arg not a widget

```
      mat[i] = atof (s);
```

*E241* 1 float = atof(float) e.g. mat[0] = atof(mat[0]) or similar nonsense
*P242* 1 ftoa called instead of atof
*T243* 0 mat[i] = mw[i] or atof (mw[i]) or similar
*O244* 1 first arg not mw[i]

```
    }
    det = mat[0]*mat[3] - mat[1]*mat[2];
    if (det != 0) {
```

*G251* 1 'if' clause is missing

```
      new[0] = mat[3]/det; new[1] = -mat[1]/det;
      new[2] = -mat[2]/det; new[3] = mat[0]/det;
```

*26x*

```
      for (i = 0; i <= 3; i++)
```

```
        XtSetStringValue (mw[i], XmCvalue, ftoa (new[i], 8, 2));
P271   7     XtSetFloatValue or XtSetIntValue called
V272   3     number format not 8,2
TZ273  0/2   sprintf used with incorrect target s
MY274  1/0   arg 2 and/or 3 of ftoa missing
TZ275  1/0   String* instead of String as third arg
O276   3     used mat[i] instead of new[i]
TZ277  0/5   no XmStringResourceName
O278   0     not 'XmCvalue'
TZ279  0/1   first arg not a widget
TZ372  0     strcpy(mw[i], ftoa(new[i], 8, 2)) or similar
T373   1     mw[i] = new[i] or ftoa(new[i]...) or similar
O374   1     first arg not mw[i]
    }
    else
      matrixErrorMessage ("Matrix cannot be inverted", mat, 8, 2);
V281   2     completely different message text
V282   7     number format not 8,2
P283   1     used CreateErrorDialog instead of matrixErrorMessage
P284   2     used printf instead of matrixErrorMessage
TZ285  1/2   not 'mat' or 'new'
O286   1     'new' instead of 'mat'
MY287  1     one or several args are missing
  }
}
```

Summing up, there are 23 errors that appear 7 or more times. The most frequent of those (12 or more times) are P234, V146, O142, T231, T212, Z162, Z152, V127, and V126.

## A.2. Program Template

The following program template was given to the experimental subjects as the starting point for problem A.

```
/* Program skeleton for Problem A (2x2 matrix inversion) */

#include <stdio.h>
#include <stdlib.h>
#include "stdmotif.h"

#ifdef __STDC__     /* this function declaration is valid in ANSI C: */
void button_pushed (Widget widget, XtPointer client_data,
                    XtPointer call_data);
#else               /* and this one in Kernighan/Ritchie-C: */
void button_pushed ();
#endif

Widget mw[4];  /* text fields for matrix coefficients:
                  0,1,2,3 for a,b,c,d */
```

```
/************************ MAIN PROGRAM *************************/

#ifdef __STDC__     /* this function declaration is valid in ANSI C: */
int main (int argc, char *argv[])
#else               /* and this one in Kernighan/Ritchie-C: */
int main (argc, argv)
  int argc;
  char *argv[];
#endif
{
  Widget        toplevel, /* main window */
                manager,  /* manager for square and buttons */
                square,   /* manager for 4 TextFields */
                buttons,  /* manager for 2 PushButtons */
                invert,   /* PushButton */
                quit;     /* PushButton */
  XtAppContext  app;
  XmString  invertlabel, quitlabel;
  int           i;

  /*---------- 1. initialize X and Motif -------------------------------*/
  /* (already complete, should not be changed) */
  globalInitialize ("A");
  toplevel = XtVaAppInitialize (&app, "Hello", NULL, 0,
        &argc, argv, fallbacks, NULL);

  /*---------- 2. create and configure widgets -------------------------*/

  /* Create vertical RowColumn manager 'manager' w. parent 'toplevel'; */
  /* Create horizontal 2-column RowColumn manager 'square' with  */
  /*   Equalization and parent 'manager'; */
  /* Create horizontal RowColumn manager 'buttons' w. parent 'manager'; */
  /* Create TextField widgets mw[0..3] with 100 pixel width each */
  /*   for coefficients a, b, c, d of the matrix; */
  /* Create PushButton 'invert' with label "Invert matrix" and */
  /*   parent 'buttons'; */
  /* Create PushButton 'quit' with label "Quit"; */

  /*---------- 3. register callback functions --------------------------*/

  /* Register callback function 'button_pushed' for the 'invert' Button */
  /*   with the number 1 as 'client_data'; */
  /* Register callback function 'button_pushed' for the 'quit' Button */
  /*   with the number 99 as 'client_data'; */

  /*---------- 4. realize widgets and turn control to X event loop -----*/
  /* (already complete, should not be changed) */
  XtRealizeWidget (toplevel);
  XtAppMainLoop (app);
  return (0);
}
```

```
/************************ Functions ************************/

#ifdef __STDC__     /* this function declaration is valid in ANSI C: */
void button_pushed (Widget widget, XtPointer client_data,
                    XtPointer call_data)
#else               /* and this one in Kernighan/Ritchie-C: */
void button_pushed (widget, client_data, call_data)
  Widget widget;
  XtPointer client_data;
  XtPointer call_data;
#endif
{
  /* this is the callback function to be called when clicking on
     the PushButtons occurs */
  double mat[4], new[4], /* old and new matrix coefficients */
         det;            /* determinant */
  String s;
  int    i;

  /* IF client_data indicates button ’quit’ */
  /* THEN */
  /*    call ’exit’ */
  /* ELSIF client_data indicates Button ’invert’ */
  /* THEN */
  /*    Read values mat[0] to mat[3] from widgets mw[0] to mw[3]; */
  /*    Compute determinant  det = mat[0]*mat[3] - mat[1]*mat[2]; */
  /*    IF determinant not equals zero */
  /*    THEN */
  /*       Compute coefficients new[0] to new[3] of the */
  /*          inverted matrix as */
  /*          new[0] = mat[3]/det; new[1] = -mat[1]/det; */
  /*          new[2] = -mat[2]/det; new[3] = mat[0]/det; */
  /*       Enter new coefficients into widgets mw[0] to mw[3] with */
  /*          8 digits width (2 decimal places); */
  /*    ELSE */
  /*       Print "Matrix cannot be inverted" by matrixErrorMessage; */
  /*    FI */
  /* FI */
}
```

# Appendix B. Problem B

## B.1. Error Numbers and Gold Program

The following list shows the body of the gold program and the error descriptions for problem B. The structure is the same as in Appendix A.1.

```
int main (argc, argv)
  int argc;
  char *argv[];
{
  Widget        main_w, menubar, menu, label;
  XtAppContext  app;

  /********** 1. initialize X and Motif ********************************/
  /* (already complete, need not be changed) */
  toplevel = XtVaAppInitialize (&app, "Hello", NULL, 0,
        &argc, argv, fallbacks, NULL);
```

*O501* 0 local 'toplevel' declared (hides global one needed in handle_menu)

```
  /********** 2. create and configure widgets **************************/

  main_w = XmCreateMainWindowWidget ("main_window", toplevel);
```

*TZ511* 1/1 1st arg not String
*TZ512* 0/0 2nd arg not Widget
*P513* 1 called XmCreateRowColumnManager...
*O515* 0 not 'toplevel'
*D516* 1 nonexisting procedure called
*O517* 3 result not assigned at all or assigned to wrong object

```
  menubar = XmCreateTrivialMenuBar (main_w, "FileBrowser",
                        XmStringCreate ("File Browser", "LARGE"), 'F');
```

*TZ521* 0/1 1st arg not Widget
*TZ522* 1/0 2nd arg not String
*TZ523* 5/3 3rd arg not XmString
*TZ524* 4/3 4th arg not char
*O525* 0 not 'main_w'
*V526* 5 XmStringCreateLocalized or not "LARGE"
*O527* 1 result not assigned at all or assigned to wrong object
*V528* 0 unreasonable value for menubar title text
*V529* 17 unreasonable value for shortcut key
*P531* 1 other procedure called
*TZ532* 1/1 wrong or missing arguments to XmStringCreate or related calls
*A533* 1 additional argument(s) to XmCreateTrivialMenuBar

```
  label = XmCreateLabelWidget ("by", main_w,
                        XmStringCreate ("by Lutz Prechelt", "SMALL"));
```

*TZ541* 0/2 1st arg not String
*TZ542* 1/2 2nd arg not Widget
*TZ543* 5/2 3rd arg not XmString
*P544* 1 called XmCreateTextFieldWidget
*O545* 7 not 'main_w' nor 'toplevel'
*V546* 3 XmStringCreateLocalized or not "SMALL"

*O547* 1 result not assigned at all or assigned to wrong object
*TZ548* 1/1 wrong or missing arguments to XmStringCreate
*P549* 1 label = XmStringCreate(...)
*AX551* 1 additional argument(s) to XmCreateLabelWidget

```
  XtSetWidgetValue (main_w, XmCworkWindow, label);
```

*TZ561* 0/0 1st arg not Widget
*TZ562* 2/4 2nd arg not XmWidgetResourceName
*TZ563* 2/1 3rd arg not Widget
*V564* 0 not 'XmCworkWindow'
*O565* 1 not 'main_w'
*O566* 0 not 'label'
*P567* 2 XtSetxxValue for wrong xx
*P568* 0 yet another procedure
*G569* 0 call is missing
*T571* 4 direct assignment to XmCworkWindow attempted

```
  menu = XmCreatePulldownMenu3 ("TheMenu", menubar, 0,
                  XmStringCreateLocalized ("Select file"), 'f',
                  XmStringCreateLocalized ("Open selected file"), 'O',
                  XmStringCreateLocalized ("Quit"), 'Q',
                  handle_menu);
```

*TZ581* 1/0 1st arg not String
*TZ582* 0/0 2nd arg not Widget
*TZ583* 0/0 3rd arg not int
*TZ584* 4/7 4th, 6th, or 8th arg not XmString
*TZ585* 2/2 5th, 7th, or 9th arg not char
*TZ586* 0/0 10th arg not function pointer
*MY587* 1/1 1st or 2nd arg missing
*MY588* 2/2 3rd arg missing
*MY589* 0/0 4th, 6th, or 8th arg missing
*MY591* 1/4 5th, 7th, or 9th arg missing
*MY592* 0/1 10th arg missing
*AZ593* 0 additional arg(s) present
*O594* 1 not 'menubar'
*V595* 9 not '0'
*V596* 0 not 'handle_menu'
*T597* 2 tried to supply prototype for callback function
*T598* 7 tried to supply arguments or () for callback function
*TZ599* 0/0 applied wrong cast or * or similar to callback function
*P601* 17 called XmStringCreate instead of XmStringCreateLocalized
*V602* 1 unreasonable value for menu entry text (4th, 6th, or 8th arg)
*V603* 3 unreasonable value for shortcut key (5th, 7th, or 9th arg)
*O604* 6 result not assigned at all or assigned to wrong object [minor]
*TZ605* 3/3 missing or wrong arg to XmStringCreate
*C606* 1 calls XtAddCallbackF
*TZ607* 1/0 wrong arguments to XtAddCallbackF
*61x*

```
  /********** 4. realize widgets and turn control to X event loop ******/
  /* (already complete, need not be changed) */
  XtRealizeWidget (toplevel);
  XtAppMainLoop (app);
  return (0);
}
```

```
/************************ Functions ************************/

void handle_menu (Widget widget, XtPointer client_data, XtPointer call_data)
{
  Widget fs;
```
*O621* 0 local redeclaration of 'toplevel'
*O622* 0 local declaration of 'main_w', 'menu', or 'menubar'
```
  if ((int)client_data == 0) {  /* first menu entry selected */
```
*V631* 0 'if' (or 'switch') query not for values 0, 1, 2 or values mixed up
*T632* 0 missing cast
*E633* 0 client_data wrongly dereferenced
*C634* 0 recursive call to 'button_pushed' or call to 'cbf'
*E635* 0 = instead of == in any 'if'
*O636* 0 'call_data' used instead of 'client_data'
*G637* 0 (if 'switch' is used:) one or more 'break' statements is missing
*64x*
```
    Widget fs = XmCreateFileSelectorDialog (toplevel, "fileselection");
```
*TZ651* 0/1 1st arg not Widget
*TZ652* 0/0 2nd arg not String
*O653* 22 not 'toplevel'
*O654* 2 result not assigned at all or assigned to wrong object
*P655* 1 other function called
*D656* 1 nonexisting function called
*TZ657* 0/0 1st and 2nd arg exchanged
*66x*
```
    XtAddCallbackF (fs, XmCokCallback, keepSelectedFile, NULL);
```
*TZ671* 1/0 1st arg not Widget
*TZ672* 0/1 2nd arg not XmFuncResourceName
*TZ673* 1/0 3rd arg not callback function pointer
*TZ674* 2/1 4th arg not NULL or XtPointer
*O675* 4 not 'fs'
*O676* 0 not 'keepSelectedFile'
*V677* 0 not 'XmCokCallback'
*P678* 13 called XtSetxxValue or other function instead
*TZ679* 5/6 wrong arguments to XtSetxxValue or other function
*TZ681* 0/0 tried to supply prototype for callback function
*TZ682* 0/0 tried to supply arguments or () for callback function
*TZ683* 0/0 applied wrong cast or * or similar to callback function
*T684* 1 XmCokCallback = keepSelectedFile or similar
*69x*
```
    XtManageChild (fs);
```
*TZ701* 0/1 1st arg not Widget
*O702* 0 not 'fs'
*P703* 1 other function called
*D704* 2 nonexisting function called
*G705* 8 call is missing
```
  }
  else if ((int)client_data == 1) {  /* second menu entry selected */
    Widget scrolltext = XmCreateScrolledTextWindow (selectedFile(), toplevel,
                                                    25, 80);
```
*TZ711* 1/4 1st arg not String
*TZ712* 0/0 2nd arg not Widget

*TZ713* 1/0 3rd or 4th arg not int
*MY714* 0/0 1st or 2nd arg is missing
*MY715* 0/0 3rd or 4th arg is missing
*V716* 23 not 'selectedFile()' or equivalent
*O717* 2 not 'toplevel'
*V718* 1 not 25, 80
*O719* 1 result not assigned at all or assigned to wrong object
*O721* 0 assigned to a Widget that is already in use and is overwritten
*P723* 1 other function called
*D724* 0 nonexisting function called
*TZ725* 1/0 uses XtSetxxValue with wrong xx or with parameter type errors
*73x*

```
    XtSetStringValue (scrolltext, XmCvalue, readWholeFile (selectedFile()));
```

*TZ741* 0/1 1st arg not Widget
*TZ742* 0/1 2nd arg not XmStringResourceName
*TZ743* 2/3 3rd arg not String
*V744* 0 not 'XmCvalue'
*O745* 2 not 'scrolltext'
*O746* 6 3rd value not contents of file
*P747* 1 XtSetxxValue for wrong xx
*P748* 1 yet another procedure
*C749* 1 some additional call to XtSetxxValue
*TZ751* 2/3 no string arg to 'readWholeFile'
*V752* 0 not 'selectedFile()'
*T753* 1 XmCvalue = readWholeFile(...) or similar

```
  }
  else if ((int)client_data == 2) {  /* third menu entry selected */
    exit (0);
```

*E761* 2 exit; (no argument and no parentheses) [exit() is acceptable]

```
  }
}
```

Summing up, there are 10 errors that appear 7 or more times. The most frequent of those (12 or more times) are V716, O653, P601, V529, and P678.

## B.2. Program Template

The following program template was given to the experimental subjects as the starting point for problem A.

```
/* Program skeleton for Problem B (File Browser) */

#include <stdio.h>
#include <stdlib.h>
#include "stdmotif.h"

#ifdef __STDC__      /* this function declaration is valid in ANSI C: */
void handle_menu (Widget widget, XtPointer client_data,
                  XtPointer call_data);
```

```
#else               /* and this one in Kernighan/Ritchie-C: */
void handle_menu ();
#endif

Widget toplevel;

/************************ MAIN PROGRAM ************************/

#ifdef __STDC__     /* this function declaration is valid in ANSI C: */
int main (int argc, char *argv[])i
#else               /* and this one in Kernighan/Ritchie-C: */
int main (argc, argv)
  int argc;
  char *argv[];
#endif
{
  Widget        main_w,  /* main window */
                menubar, /* the one-entry menu bar */
                menu,    /* the pulldown menu */
                label;   /* the label displayed in the work window */
  XtAppContext  app;

  /*---------- 1. initialize X and Motif ------------------------------*/
  /* (already complete, should not be changed) */
  globalInitialize ("B");
  toplevel = XtVaAppInitialize (&app, "Hello", NULL, 0,
        &argc, argv, fallbacks, NULL);

  /*---------- 2. create and configure widgets ------------------------*/

  /* Create MainWindow widget 'main_w' with parent 'toplevel'; */
  /* Create MenuBar 'menubar' w. entry 'File Browser' in 'LARGE' font */
  /*   and with parent 'main_w'; */
  /* Create Label widget 'label' w. entry 'by <Name>' in 'SMALL' font; */
  /* Enter 'label' as 'workWindow' of 'main_w'; */
  /* Create PulldownMenu 'menu' with entries 'Select file', 'Open */
  /*   selected file' and 'Quit' (callback function 'handle_menu'); */

  /*---------- 3. register callback functions -------------------------*/
  /* (handle_menu was already registered above, nothing to be done) */

  /*---------- 4. realize widgets and turn control to X event loop ----*/
  /* (already complete, should not be changed) */
  XtRealizeWidget (toplevel);
  XtAppMainLoop (app);
  return (0);
}

/************************ Functions ***********************/

#ifdef __STDC__     /* this function declaration is valid in ANSI C: */
void handle_menu (Widget widget, XtPointer client_data,
                  XtPointer call_data)
```

```
#else                /* and this one in Kernighan/Ritchie-C: */
void handle_menu (widget, client_data, call_data)
  Widget widget;
  XtPointer client_data;
  XtPointer call_data;
#endif
{
  Widget fs;
  if ((int)client_data == 0) {  /**** first menu entry selected ****/
    Widget fs;
    /* Create and manage FileSelectorDialog 'fs'; */
    /* Enter 'keepSelectedFile' as callback function of 'OK' button; */
  }
  else if ((int)client_data == 1) {  /**** second entry selected ****/
    Widget scrolltext;
    /* Create 25 x 80 character ScrolledTextWindow 'scrolltext' with */
    /*   parent 'toplevel' and the selected file name as the title; */
    /* Read the selected file and enter its contents into 'value' */
    /*   resource of 'scrolltext'; */
  }
  else if ((int)client_data == 2) {  /**** third entry selected ****/
    /* call 'exit'; */
  }
}
```

# Appendix C. Error Categories and Code Letters

The following list defines which error categories (as discussed in the text in Section 3.4.) were assigned to the numbers shown in the previous sections.

- The following error numbers belong to category *non-error*: 151 161 283.
- The following error numbers belong to category *comp*: 302 303 124 312 313 322 323 147 332 333 336 157 352 167 362 171 191 192 193 194 195 181 201 202 203 204 205 212 213 214 222 223 225 233 242 516 549 571 597 598 599 632 634 635 656 681 682 683 684 704 724 753.
- The following error numbers belong to category *invis*: 122 311 565 568 569.
- The following error numbers belong to category *slight*: 351 361 272 281 282 529 601 602 603 718.
- The following error numbers belong to category *severe*: 101 102 112 113 115 116 117 301 123 125 126 127 128 132 133 135 136 137 321 142 143 145 146 331 335 155 156 159 165 166 169 175 176 177 178 185 186 187 188 211 215 216 341 221 226 237 244 251 273 276 278 374 284 286 501 513 515 517 525 526 527 528 531 544 545 546 547 564 566 594 595 596 604 606 621 622 631 633 636 637 653 654 655 675 676 677 702 703 705 716 717 719 721 723 744 745 746 748 749 752.
- The following error numbers belong to category *severeD*: 111 304 121 314 315 131 324 141 148 334 152 153 353 162 163 363 172 173 174 179 196 182 183 184 189 206 231 232 234 235 236 238 239 241 243 271 274 275 277 279 372 373 285 287 512 521 522 523 524 532 541 542 543 548 563 581 582 583 584 585 586 587 588 589 591 592 605 607 651 657 671 672 673 678 679 701 711 712 713 714 715 725 741 742 743 751 761.
- The following error numbers belong to category *invisD*: 114 118 119 305 129 134 139 325 144 149 154 158 164 168 511 533 551 561 562 567 593 652 674 747.

The above categories describe the effect of an error and could thus be called external categories. Beside this category, each error number is also associated with an error-type code letter that describes the type or origin of the error. You find such a letter in front of each error number in the error-number lists. The purpose of the error-type codes is to make consistent application of the error numbers simpler and to facilitate further analyses of the annotations that are not described in this report.

The meaning of the error type codes is as follows:

A additional parameter(s)
X ditto, but not flagged by compiler
M missing parameter(s)
Y ditto, but not flagged by compiler
T parameter or operand of wrong type
Z ditto, but not flagged by compiler

B debug code inserted/removed/changed [is not an error]
C additional procedure call that should not be there
D missing or wrong declaration or typo in use of existing object that leads to confusion. (Not all errors of this type are annotated.)
E faulty expression [only if V does not apply]
G gap: missing statement, although program section is otherwise finished
O wrong variable object used as argument or assigned to
P wrong procedure called for desired effect (or wrong desired effect)
V wrong value given for parameter or used in expression [see E and O]

# Appendix D. The Wrapper Library

Instead of reproducing the complete library documentation given to the subjects, we will show only the interface description of the wrapper library here. The subjects were not told to look into this file, however, since the documentation contained more information and also cross-references.

This is the file `stdmotif.h`:

```
/* Simple interface module for standardized calls of Motif widgets */
/* simplifies the parameter lists. All functions have fully */
/* type-checked prototypes with ANSI C and none with K&R C */
/* Lutz Prechelt, 1995/05/31 */

#include <Xm/Xm.h>
#define __USE_FIXED_PROTOTYPES__
#include <stdio.h>
#include <stdlib.h>

#ifdef __STDC__
#define _A_(l) l
#else
#define _A_(l) ()
#endif

/*************** Motif extensions *******************************/

/* artificial type-checking for resource-name <--> resource type */
typedef struct { String a; } XmIntResourceName;
typedef struct { String a; } XmFloatResourceName;
typedef struct { String a; } XmFuncResourceName;
typedef struct { String a; } XmStringResourceName;
typedef struct { String a; } XmWidgetResourceName;


/* type-checkable resource name constants */
extern XmIntResourceName
                  XmCcolumns,
                  XmCnumColumns,
                  XmCorientation,
                  XmCpacking,
                  XmCrows,
                  XmCwidth;
extern XmFuncResourceName
                  XmCactivateCallback,
                  XmCcancelCallback,
                  XmChelpCallback,
                  XmCokCallback;
extern XmStringResourceName
                  XmCvalue;
extern XmWidgetResourceName
                  XmCworkWindow;
```

```
/* ----- create a popup dialog window widget with an OK button */
Widget XmCreateErrorDialogOK _A_((Widget parent, String resourcename,
                              XmString message));

/* ----- create file selector dialog box */
Widget XmCreateFileSelectorDialog _A_((Widget w, String name));

/* ----- create a label widget with given label text */
Widget XmCreateLabelWidget _A_((String resourcename, Widget parent,
                                XmString text));

/* ----- create a bare MainWindow widget */
Widget XmCreateMainWindowWidget _A_((String resourcename, Widget parent));

/* ----- create a pulldown menu with exactly three entries */
Widget XmCreatePulldownMenu3 _A_((String resourcename, Widget menubar,
                        int menunumber,
                        XmString entry1, char key1,
                        XmString entry2, char key2,
                        XmString entry3, char key3,
                        void (*callback) (Widget, XtPointer, XtPointer)));

/* ----- create a PushButton widget with a given label */
Widget XmCreatePushButtonL _A_((String resourcename, Widget parent,
                            XmString label));

/* ----- create a RowColumn manager */
Widget XmCreateRowColumnManager _A_((String resourcename, Widget parent));

/* ----- create a preconfigured RowColumn manager */
Widget XmCreateRowColumnManagerOCP _A_((String resourcename,
                                        Widget parent,
                                        int orientation, int numColumns,
                                        Boolean equalize));

/* ----- create a scrolled text widget without any resources set */
Widget XmCreateScrolledTextWidget _A_((Widget parent,
                                       String resourcename));

/* ----- create a stand alone scrolled text widget of lines x columns */
Widget XmCreateScrolledTextWindow _A_((String resourcename, Widget parent,
                                      int nrOfLines, int nrOfColumns));

/* ----- create empty TextField widget */
Widget XmCreateTextFieldWidget _A_((String resourcename, Widget parent));

/* ----- create TextField widget with given width and initial string */
Widget XmCreateTextFieldWidgetW _A_((String resourcename, Widget parent,
                                     int width, String value));

/* ----- create MenuBar widget with exactly one entry */
Widget XmCreateTrivialMenuBar _A_((Widget parent, String resourcename,
```

```
                                   XmString menuname, char key));

/* ----- install callback function with type-checked resource name */
void XtAddCallbackF _A_((Widget w, XmFuncResourceName r,
                     void (*f) (Widget, XtPointer, XtPointer),
                     XtPointer client_data));

/* ----- read a value of a resource from a widget */
void XtGetIntValue _A_((Widget w, XmStringResourceName r,
                        int *value));
void XtGetFloatValue _A_((Widget w, XmFloatResourceName r,
                          double *value));
void XtGetStringValue _A_((Widget w, XmStringResourceName r,
                           String *value));
void XtGetWidgetValue _A_((Widget w, XmStringResourceName r,
                           Widget *value));

/* ----- change the value of a resource for a widget */
void XtSetIntValue _A_((Widget w, XmIntResourceName r,
                        int value));
void XtSetFloatValue _A_((Widget w, XmFloatResourceName r,
                          double value));
void XtSetStringValue _A_((Widget w, XmStringResourceName r,
                           String value));
void XtSetWidgetValue _A_((Widget w, XmWidgetResourceName r,
                           Widget value));


/*************** other extensions ********************************/

/* ----- fallback resources ----- */
extern String fallbacks[];

/* ----- convert floating point number into String of given form */
String ftoa _A_((double x, int width, int precision));

/* ----- global Initialization (protocol stamp) */
void globalInitialize _A_((String programname));

/* ----- callback function: store filename selected in fileselector box */
void keepSelectedFile _A_((Widget w, XtPointer client_data,
                           XtPointer callback_data));

/* ----- print error message along with 2x2 matrix in given format */
void matrixErrorMessage _A_((String message,
                             double matrixcoefficients[4],
                             int width, int precision));

/* ----- read file and return its contents in newly allocated string */
String readWholeFile _A_((String filename));

/* ----- retrieve filename stored by keepSelectedFile() */
String selectedFile _A_(());
```

# Appendix E. Subject Data

Information about the experimental subjects: sex, experimental group, education, occupation, total programming experience $e$ in years, lines programmed in C/C++, lines programmed in X Windows or Motif. ("theor." means zero practical programming experience, but theoretical knowledge.)

| # | sex | G | education | occupation | $e$ | C/C++ | X/Motif |
|---|---|---|---|---|---|---|---|
| 003 | m | 2 | M.S. CS | Ph.D. student CS | 10 | <30000 | 0 |
| 005 | m | 2 | B.S. CS | M.S. student CS | 10 | <30000 | 0 |
| 006 | m | 3 | B.S. CS | M.S. student CS | 9 | <3000 | 0 |
| 007 | m | 4 | B.S. CS | M.S. student CS | 10 | <30000 | 0 |
| 008 | m | 1 | B.S. CS | M.S. student CS | 7 | <3000 | theor. |
| 009 | m | 2 | Ph.D. CS | Postdoc CS | 15 | >30000 | 0 |
| 010 | m | 3 | B.S. CS | M.S. student CS | 7 | <3000 | 0 |
| 011 | m | 4 | M.S. CS | Ph.D. student CS | 19 | >30000 | <30000 |
| 012 | m | 1 | M.S. CS | Sys.progr.&admin. | 15 | <30000 | <3000 |
| 013 | f | 2 | M.S. Physics | Ph.D. student CS | 8 | >30000 | <300 |
| 014 | m | 3 | M.S. CS | Ph.D. student CS | 8 | <3000 | 0 |
| 015 | m | 4 | M.S. CS | Ph.D. student CS | 17 | >30000 | 0 |
| 016 | m | 1 | M.S. CS | Ph.D. student CS | 10 | <300 | 0 |
| 017 | m | 2 | B.S. CS | M.S. student CS | 10 | >30000 | 0 |
| 018 | m | 3 | B.S. CS | M.S. student CS | 9 | <30000 | 0 |
| 019 | m | 4 | M.S. CS | Ph.D. student CS | 12 | >30000 | 0 |
| 020 | m | 1 | Ph.D. CS | Postdoc CS | 15 | <30000 | <300 |
| 021 | m | 2 | B.S. CS | M.S. student CS | 5 | <3000 | 0 |
| 023 | m | 4 | M.S. CS | Ph.D. student CS | 7 | >30000 | 0 |
| 025 | m | 2 | B.S. CS | M.S. student CS | 6 | <30000 | <300 |
| 026 | m | 3 | M.S. CS | Ph.D. student CS | 6 | >30000 | 0 |
| 027 | m | 4 | B.S. CS | M.S. student CS | 8 | <3000 | 0 |
| 028 | m | 1 | M.S. CS | Ph.D. student CS | 12 | <30000 | theor. |
| 029 | m | 2 | M.S. CS | Ph.D. student CS | 10 | <30000 | <300 |
| 031 | m | 4 | B.S. CS | M.S. student CS | 12 | <30000 | 0 |
| 032 | f | 1 | M.S. CS | Ph.D. student CS | 10 | <3000 | 0 |
| 033 | m | 2 | M.S. CS | Ph.D. student CS | 12 | <3000 | 0 |
| 036 | m | 1 | M.S. CS | Ph.D. student CS | 15 | >30000 | <3000 |
| 037 | m | 2 | B.S. CS | M.S. student CS | 10 | <3000 | 0 |
| 038 | m | 3 | M.S. CS | Ph.D. student CS | 10 | <30000 | <300 |
| 039 | m | 4 | M.S. CS | Ph.D. student CS | (questionnaire lost) | | |
| 040 | f | 1 | M.S. CS | Ph.D. student CS | 4 | <3000 | 0 |
| 041 | m | 2 | M.S. CS | Ph.D. student CS | 13 | <30000 | 0 |
| 042 | m | 3 | M.S. CS | Ph.D. student CS | 8 | >30000 | <300 |

# References

[Basili and Perricone 1984] Victor R. Basili and B.T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–52, January 1984.

[Ebrahimi 1994] Alireza Ebrahimi. Novice programmer errors: Language constructs and plan composition. *Intl. J. of Human-Computer Studies*, 41:457–480, 1994.

[Frankl and Weiss 1993] Phyllis G. Frankl and Stewart N. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Trans. on Software Engineering*, 19(8):774–787, August 1993.

[Hudak and Jones 1994] Paul Hudak and Mark P. Jones. Haskell vs. Ada vs. C++ vs. awk vs. . . . an experiment in software prototyping productivity. Technical report, Yale University, Dept. of CS, New Haven, CT, July 1994.

[Humphrey 1995] Watts Humphrey. *A Discipline for Software Engineering*. SEI Series in Software Engineering. Addison Wesley, Reading, MA, 1995.

[Soloway and Iyengar 1986] Elliot Soloway and Sitharama Iyengar, editors. *Empirical Studies of Programmers*. Ablex Publishing Corp., Norwood, NJ, June 1986. (The papers of the First Workshop on Empirical Studies of Programmers, Washington D.C.)

[Spohrer and Soloway 1986] James G. Spohrer and Elliot Soloway. Analyzing the high frequency bugs in novice programs. In *[Soloway and Iyengar 1986]*, pages 230–251, 1986.

[Wirth 1994] Nikolaus Wirth. Gedanken zur Software-Explosion. *Informatik Spektrum*, 17(1):5–20, February 1994.

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b. RESTRICTIVE MARKINGS<br>None |
| 2a. SECURITY CLASSIFICATION AUTHORITY<br>N/A | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for Public Release<br>Distribution Unlimited |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE<br>N/A | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>CMU/SEI-96-TR-014 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>ESC-TR-96-014 |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Software Engineering Institute<br>6b. OFFICE SYMBOL (if applicable)<br>SEI | 7a. NAME OF MONITORING ORGANIZATION<br>SEI Joint Program Office |
| 6c. ADDRESS (city, state, and zip code)<br>Carnegie Mellon University<br>Pittsburgh PA 15213 | 7b. ADDRESS (city, state, and zip code)<br>HQ ESC/ENS<br>5 Eglin Street<br>Hanscom AFB, MA 01731-2116 |
| 8a. NAME OFFUNDING/SPONSORING ORGANIZATION<br>SEI Joint Program Office<br>8b. OFFICE SYMBOL (if applicable)<br>ESC/ENS | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>F19628-95-C-0003 |
| 8c. ADDRESS (city, state, and zip code))<br>Carnegie Mellon University<br>Pittsburgh PA 15213 | 10. SOURCE OF FUNDING NOS.<br>PROGRAM ELEMENT NO: 63756E<br>PROJECT NO.: N/A<br>TASK NO: N/A<br>WORK UNIT NO.: N/A |

11. TITLE (Include Security Classification)

A Controlled Experiment Measuring the Effect of Procedure Argument Type Checking on Programmer Productivity

12. PERSONAL AUTHOR(S)

Lutz Prechelt, Walter F. Tichy

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (year, month, day) | 15. PAGE COUNT |
|---|---|---|---|
| Final | FROM TO | 1996, June | 41 |

16. SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB. GR. |
| | | |
| | | |
| | | |

18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)

type checking, experiment, errors, defects, ANSI C

19. ABSTRACT (continue on reverse if necessary and identify by block number)


Type checking is considered an important mechanism for detecting programming errors, especially interface errors. This report describes an experiment to assess the error-detection capabilities of static intermodule type checking.

The experiment uses ANSI C and Kernighan&Ritchie (K&R) C. The relevant difference is that the ANSI C compiler checks module interfaces (i.e., the parameter lists of calls to external functions), whereas K&R C does not. The experiment employs a counterbalanced design in which each subject writes two non-trivial programs that interface with a complex library (Motif). Each subject writes one program in ANSI C and one in K&R C. The input to each compiler run is saved and manually analyzed for errors.

Results indicate that delivered ANSI C programs contain significantly fewer interface errors than delivered K&R C programs. Furthermore, after subjects have gained some familiarity with the interface they are using, ANSI C programmers remove errors faster and are more productive (measured in both time to completion and functionality implemented).

This report describes the design, setup, and results of the experiment including complete source code and error lists.


(please turn over)

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■ | 21. ABSTRACT SECURITY CLASSIFICATION<br>Unclassified, Unlimited Distribution |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Thomas R. Miller, Lt Col, USAF | 22b. TELEPHONE NUMBER (include area code)<br>(412) 268-7631<br>22c. OFFICE SYMBOL<br>ESC/ENS (SEI) |