ELSEVIER

Information and Software Technology 38 (1996) 477–492

INFORMATION AND SOFTWARE TECHNOLOGY

Programmers' performance on structured versus nonstructured function definitions

K.G. van den Berg*, P.M. van den Broek

University of Twente, Faculty of Computer Science, P.O. Box 217, 7500 AE Enschede, The Netherlands

Received 22 February 1995; revised 20 September 1995; accepted 26 September 1995

Abstract

A control-flow model for functional programs is used in an experimental comparison of the performance of programmers on structured versus nonstructured Miranda function definitions. The performance is taken as a measure of the comprehensibility of functional programs. The experimental set-up is similar to the Scanlan study (1989). However, in the present study, a two-factor repeated measures design is used in the statistical analysis. The control-flow model appears to be useful in the shaping of the experiment. A significantly better performance has been found for structured function definitions on both dependent variables: the time needed to answer questions about the function definitions and the proportion correct answers. Moreover, for structured function definitions, a counter-intuitive result has been obtained: there are significantly fewer errors in larger definitions than in smaller ones.

Keywords: Structured programming; Functional programming; Software metrics; Comprehensibility

1. Introduction

There is a long-standing discussion on structured programming in the literature (e.g. the survey of Vessey and Weber [1]). Most of this research has been carried out in the domain of imperative programming. This paper will present an experiment on programmers' performance in the domain of functional programming for structured versus nonstructured function definitions. The set-up of this experiment is similar to the Scanlan [2] study in his comparison of structured flowcharts and pseudocode. However, our experimental design and statistical analysis are different from this study: these differences will be explained in the subsequent sections.

The characterization of the structure of function definitions is based on a control-flow model as defined in a previous paper [3]. The framework for experimentation in software engineering [4] will be used in the following outline of the present study.

The *motivation* of this study has been indicated above: an experimental comparison of programmers' performance on structured versus nonstructured 'programs' in the domain of functional programming. As such, it can be seen as a contribution to the discussion on structured programming mentioned above. The actual *objects* in this study are Miranda[1] function definitions [5]. The attribute structure of function definitions will be defined in terms of a control-flow model [3]. The control-flow in function definitions is determined by patterns and guards, as will be described in following sections. Based on this model it is possible to distinguish structured and nonstructured function definitions.

The *purpose* of this study is to validate empirically some programming style rules on the use of guards and patterns in function definitions with respect to programmers' performance. This performance is taken as a measure of the comprehensibility of functional programs.

> Pattern matching is one of the cornerstones of an equational style of definition; more often than not it leads to a cleaner and more readily understandable definition than a style based on conditional equations [with guards] [6].

The *hypothesis* to be tested in the experiments is that programmers perform better on structured function

* email:vdberg@cs.utwente.nl.

[1] Miranda is a trademark of Software Research Limited.

SSDI 0950-5849(95)01085-8

definitions than on comparable nonstructured definitions, and hence that programs with structured function definitions are easier to comprehend than programs with comparable nonstructured definitions.

One *perspective* in this study on programmers' performance is that of a formal technical review of coding or a code walkthrough: i.e. inspection of code written by another programmer [7]. Programmers have to comprehend the code and make statements about the behaviour of the program. The *domain* of this study can be characterized as programming-in-the-small by novice programmers (Computer Science students). The *scope* of the study is that of a single programmer working on a single program-unit (a Miranda function definition).

In the following section, patterns and guards in function definitions are described in more detail, succeeded with a recapitulation of the control-flow model and the description of structured versus nonstructured function definitions. In subsequent sections, the experiment will be described, with the results and followed by a discussion and some conclusions.

2. Function definitions

A description of patterns and guards in Miranda function definitions is given in Peyton Jones [8]. An example is given below: a definition of the function *split* (the line numbers have been added).

```
split : : (* → bool) → [*] → ([*], [*])          1
split p []                                        2
   = ([], [])                                     3
split p (x : xs)                                  4
   = (x : ys, zs), if p x                         5
   = (ys, x : zs), if ~ (p x)                     6
      where (ys, zs) = split p xs                 7
```

The function *split* returns, for given a predicate, i.e. a boolean function with type (* → bool), and a list with type [*], a tuple with two components: the first component is the list with elements satisfying the predicate and the second component is the list with elements not satisfying the predicate. In line 1, the type of the function *split* is given: it is a polymorphic function (a star * denotes a type variable). For example, evaluation of the expression *split even [2,4,7,4]* yields the tuple *([2,4,4],[7])*.

The second argument of the function *split* is a list. In the first clause of the definition (line 2–3), the pattern *[ ]* for an empty list is used for the selection of this clause. In the second clause (line 4–7), the non-empty list pattern *(x:xs)* is used. In the second clause there are two cases, one with the guard *p x* (line 5), the other with the guard ~ *(p x)*. In the local definition on line 7, the tuple *(xs, ys)* is defined in terms of a recursive call of *split*.

The patterns in this definition are *disjoint*: if one pattern matches, then there is no other pattern that will match. E.g., if the actual argument list matches the pattern *[ ]* then no other pattern will match, and the same applies to pattern *(x:xs)*. Moreover, these patterns are *exhaustive*: for any argument there will be a pattern that will match. E.g., a list-argument is either empty and matches the pattern *[ ]*, or it is non-empty and matches the pattern *(x:xs)*. The guards in this definition are disjoint as well: if p *x* is True then no other guard is True; moreover, these guards are exhaustive: either *p x* is True or ~*(p x)* is True.

In this definition of *split*, the meaning of the definition is independent of the *textual order* of the clauses and cases. However, quite commonly, the meaning depends on the order of the clauses and the cases. Moreover, guards in Miranda function definitions may interact with pattern matching. There are few examples in the literature to demonstrate the latter. In the first example, the function *funnyLastElt* returns the first negative element of its argument list, or if there is no such element, it returns the last element of this list [8]:

```
funnyLastElt : : [num] → num
funnyLastElt (x : xs) = x, if x < 0
funnyLastElt (x : []) = x
funnyLastElt (x : xs) = funnyLastElt xs
```

If an argument list is not empty, then the first clause is selected and the guard $x < 0$ will be evaluated. If this condition is True, the function returns *x*; otherwise (because there is no other guard), the following pattern *(x:[ ])* will be checked, and so on. The meaning of the definition depends on the order of the clauses: e.g. if one exchanges the first clause with the last, the function does not satisfy the given specification anymore.

Another example is given by Holyer [9] with an (unusual) definition of the Miranda standard function *drop*:

```
drop : : num → [*] → [*]
drop n xs          = error "fractional",
                       if ~ integer n
                   = xs, if n ⩽ 0xs = []
drop (n + 1) (x : xs) = drop n xs
```

The function is specified as follows: *drop n xs* removes the first *n* elements from the argument list *xs*; if *n* is not an integer, there will be a program error; if *n* is negative the argument list will be returned.

In the second clause of this definition, there is a matching on the list pattern *(x:xs)* and the integer pattern

$(n+1)$. The meaning of the definition depends on the order of the clauses as well as of the order of the guards.

The interaction of patterns and guards implies that there have to be rules about the *operational semantics*. As stated above, for Miranda these rules are: patterns in a clause are evaluated from left to right, and guards in textural order; and clauses are evaluated in textual order [8]. Obviously, there is an operational bias in the language design in Miranda [10]. The control-flow model [3] captures the operational semantics of function definitions.

Some *programming style rules* with respect to use of patterns and guards in function definitions can be found in the literature, each with a different strength:

(1) Use total function definitions [11] (both exhaustive patterns and exhaustive guards).
(2) Use order independent clauses in function definition [6,8].
(3) Use exhaustive patterns [6].
(4) Use disjoint patterns [9].
(5) Use order independent alternatives for each clause [6].
(6) Use exhaustive guards [11].
(7) Use disjoint guards [11].

One of such rules is the subject of experimental validation as will be described in the subsequent sections. First, the control-flow model will be recapitulated.

3. Control-flow model

Flowgraphs are used for the modelling of control-flow in imperative programs [12]. The nodes in the directed graphs correspond to statements in the programs, whereas the edges from one node to the other indicate a flow of control between corresponding statements. The stop node in a flowgraph has outdegree zero, and every node lies on some path from the start node to the stop node. The nodes with outdegree equal to 1 are called procedure nodes; all other nodes are termed predicate nodes. For example, an elementary action is modelled as flowgraph P_1 in Fig. 1a; the if-then construct in a program is modelled as flowgraph D_0 in Fig. 1b; the if-then-else construct is modelled as flowgraph D_1 in Fig. 1c.

Flowgraphs can be concatenated (sequencing) to a new flowgraph; and flowgraphs can be nested on each other. An example of nesting D_0 onto D_1 at node 6 in Fig. 1c, is given in Fig. 1d. This is denoted as $D_1(D_0)$, in which is abstracted from the node onto which is nested. Associated with any flowgraph is a decomposition tree which describes how the flowgraph is built by sequencing and nesting elementary flowgraphs, such as D_0 and D_1. The decomposition tree of the flowgraph in Fig. 1d is depicted in Fig. 1e.

The operational semantics of Miranda function definitions is captured in the control-flow model [3]. For example, the control-flow graph for the function definition *split* is given in Fig. 2.

The four vertical lines indicate the kind of nodes in these flowgraphs: predicate nodes (outdegree 2) for patterns and guards, procedure nodes (outdegree 1) for the expressions, and finally the stop node (outdegree 0). For the predicate nodes, the True (T) and False (F) branches are indicated. Note that the lower (False) branch starting at the pattern *(x:xs)* is infeasible because either the pattern *[ ]* or the pattern *(x:xs)* will succeed: these two patterns are exhaustive. The same applies to the lower (False) branch starting at the guard $\sim$(p x): in any case, one of these guards will have the value True. However, in this model is abstracted from the actual content of the patterns and guards.

Flowgraphs can be uniquely decomposed into a hierarchy of (indecomposable) prime flowgraphs. For example, the decomposition of the flowgraph given in Fig. 2 is $D_1(D_0(D_1(D_0)))$. In this case, the depth of decomposition is 4.

3.1. Structured and nonstructured function definitions

We will give some additional definitions, as will be used in the description of the experiment in the following section. A *path* in a flowgraph is a sequence of consecutive nodes from the start node to the stop node. A *D-structured* path is given by a sequence of patterns followed by a

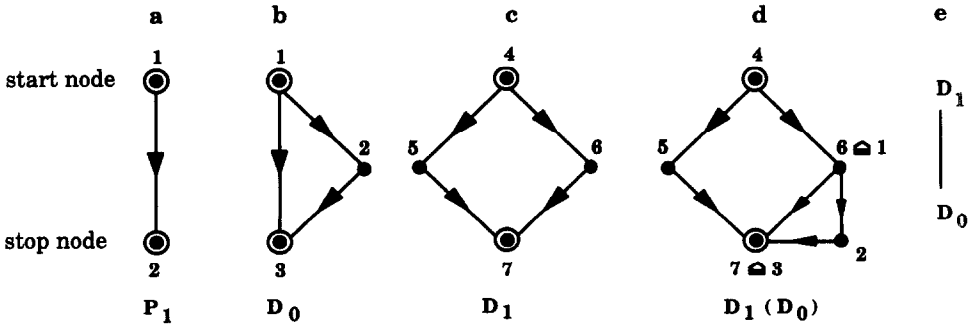

Fig. 1. Elementary flowgraphs and decomposition tree.

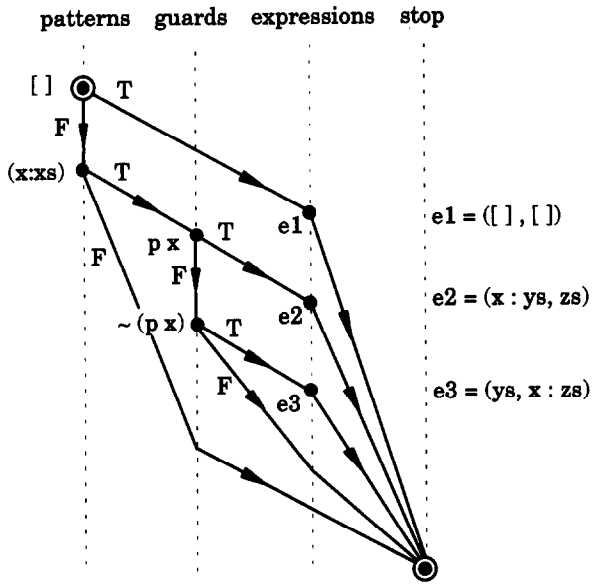

Fig. 2. Annotated control-flow graph of the function split.

sequence of guards, then possibly an expression node, and then the stop node. An *X-structured* path[2] is a path that is not D-structured: i.e., a sequence in which a guard is followed by a pattern. (In flowgraphs as drawn in Fig. 3a and Fig. 3b, an X-structured path can be identified by an edge directed from right to left). A D-structured path and an X-structured path are called *similar* if the D-structured path is a permutation of the X-structured path. A function definition is *structured (D-structured)* if all paths in its flowgraph are D-structured; otherwise the definition is *nonstructured (X-structured)*.

Function definitions are called *comparable* if their flowgraphs contains the same predicate nodes (patterns, guards), and the expression nodes represent simple numeric constants. (Comparable functions need not to be semantically equivalent.) Two example scripts[3] with comparable definitions are given in Table 1.

The flowgraphs of these function definitions are given in Fig. 3. The function f in script 101 is X-structured: it contains an X-structured path (with the edge from the guard $x > 2$ to the pattern (x:xs)). The function f in script 103 is D-structured: it contains only D-structured paths.

The evaluation of *top = f [1,2,3]* in script 101 results in an X-structured path with the following sequence of nodes (length of path = 6):

⟨start node, (x : y : z : zs), x > 2, (x : xs),
2, stop node⟩

The evaluation of *top = f [3,4]* in script 103 results in a D-structured path:

⟨start node, (x : y : z : zs), (x : xs), x > 2,
2, stop node⟩

Moreover, these two paths are similar: the path in script 101 is a permutation of the path in script 103.

It should be remarked here that it is possible to devise structured and nonstructured function definitions that are semantically equivalent. As an example, consider the following definitions:

f (x : xs) = 1, if x > 2
= 2, otherwise
f [x] = 3
g (x : xs) = 1, if x > 2
= 2, if x ⩽ 2
g [x] = 3

In the control-flow model, the function definition of f is structured, whereas the semantically equivalent definition of g is nonstructured: in the model is abstracted from the actual content of the guards, ignoring the fact that also the function g has total guards.

The *hypothesis* is that programmers' performance on the D-structured path is better than on the similar X-structured path, and hence (this is our *assumption*) that structured function definitions are better than comparable nonstructured function definitions. In the subsequent section, criteria for programmers' performance are established, and an experiment is described to test this hypothesis.

Table 1
An X-structured function definition in script 101 and a comparable D-structured function definition in script 103

```
|| script 101
f :: [num] → num
f (x : y : z : zs) = 1,   if x > 2
f (x : xs)         = 2
f xs               = 3
top = f [1,2,3]
```

```
|| script 103
f :: [num] → num
f (x : y : z : zs) = 1
f (x : xs)         = 2,   if x > 2
                   = 3,   otherwise
f xs               = 4
top = f [3,4]
```

[2] X refers to a prime other than D_0 and D_1.
[3] The numbers refer to the script numbers used in the experiment (see the Appendix).

Table 2
Properties of small, medium and large function defintions

Level \ Metric	Path length	NLOC	#Nodes	McCabe's cycle. comp.	Depth of decomposition
Small	6	5–6	7–8	4	2–3
Medium	9	8–9	13–14	7	4–6
Large	12	13–14	21–22	11	6–9

4. Experiment

In this section the design of the experiment will be described. The aim of the experiment is to validate the following programming style rule: '*Use structured function definitions instead of nonstructured ones*'. In the experiment we will test the performance of programmers on structured versus comparable nonstructured function definitions. The experimental set-up is similar to the one used by Scanlan [2] in the comparison of structured flowcharts and pseudocode. However, the experimental design and the statistical analysis presented here are different. These differences will be brought up in the subsequent sections. First, we will consider the independent and dependent variables in our experiment, followed by a description of the experimental design, the statistical model and the hypotheses.

4.1. Independent variables

The two independent variables in the experiment are the following:

- The *Size* of a script with the function definition and the top expression. The three levels of *Size*, i.e. *Small*, *Medium* and *Large*, are characterized by the length of the path belonging to the top expression, the net lines of codes of the scripts[4] (NLOC), and some control-flow metrics [3] (see Table 2). Despite the relatively small number lines of code, the function definitions—especially the larger ones—are rather complex (e.g., McCabe's cyclomatic complexity number and the depth of decomposition).
- The *Structure* of the function definition in a script. The two levels of *Structure* are the nonstructured function definition (*X-structured*) and the structured function definition (*D-structured*), as described in a previous section.

4.2. Dependent variables

The basic condition in the experiment is: *no time pressure*, i.e. the subjects are allowed to spend as much time as they need to answer the questions (cf. Scanlan [2]). To quote Moher et al. [13]:

> The most basic task [in computer programming], and yet in some ways the hardest to measure, is program comprehension [13].

The dependent variables in this experiment are two criteria on programmers' performance:

- The time to answer (*Time*), i.e. the number of seconds the subjects viewed the script and spent answering the question about the script. This is a continuous random variable.
- The correctness of the answer (*Correctness*), i.e. the answer given by the subject about the script is either correct or wrong. This is a binary random variable.

In a following section, the questions about the scripts used in the experiment are described in more detail.

4.3. Experimental design

The experimental design will be considered as a two-factor design with two dependent variables and six treatments. A treatment corresponds to a combination of factor levels. The first factor is the *Structure* with two levels: *Structured* (D) and *Nonstructured* (X). The second factor is the *Size* with three levels: *Small* (S), *Medium* (M) and *Large* (L). The design has been given schematically in Table 3.

Each treatment in the design will be given to each subject (as in the Scanlan study). The subjects are viewed as a random sample from a population. This is a two-factor experiment with repeated measures on all treatments, equal sample sizes, random subject effects and fixed factor effects [14]. (This is contrary to the

Table 3
Experimental design with factors, levels and treatments

Factor	Levels	Size		
		Small S	Medium M	Large L
Structure	Nonstructured X	treatment SX	treatment MX	treatment LX
	Structured D	treatment SD	treatment MD	treatment LD

[4] The blank lines and comment lines are not counted; the scripts also contain the type of the function and the top level expression (cf. Table 1).

Table 4
Experimental design with treatment means and factor level means

Factor	Levels	Size			
		Small S	Medium M	Large L	Mean
Structure	Nonstructured X Structured D	μ_{SX} = mean for treatment SX μ_{SD} = mean for treatment SD	μ_{MX} = mean for treatment MX μ_{MD} = mean for treatment MD	μ_{LX} = mean for treatment LX μ_{LD} = mean for treatment LD	$\mu_{.X}$ = mean for factor level X $\mu_{.D}$ = mean for factor level D
		$\mu_{S.}$ = mean for factor level S	$\mu_{M.}$ = mean for factor level M	$\mu_{L.}$ = mean for factor level L	$\mu_{..}$ = overall mean

one-factor repeated-measures design as conceived by Scanlan [2]).

The number of levels for the factor *Size* is *a* (a = 3); the number for the factor *Structure* is *b* (b = 2). The treatment mean at level j of *Size* and level k of *Structure* will be denoted by μ_{jk} with $j \in \{S, M, L\}$ and $k \in \{X, D\}$.

We will use the following point notation for the factor level means:

$$\mu_{j.} = \Sigma k \mu_{jk}/b \quad \text{and} \quad \mu_{.k} = \Sigma j \mu_{jk}/b$$

The overall mean is denoted by $\mu_{..}$ with

$$\mu_{..} = \Sigma j \Sigma k \mu_{jk}/ab = \Sigma k \mu_{j.}/a = \Sigma j \mu_{.k}/b$$

The denotation for the treatment means and the factor level means are given in Table 4.

4.4. Statistical model

The following model will be used in the statistical analysis of the experimental design described in the previous section [14].

Let y_{ijk} be the observed value on the dependent random variable Y_{ijk} for subject i ($i \in [1..n]$) for the factor A (here *Size*) at level j and the factor B (here *Structure*) at level k.

Then, we assume the following *repeated measures model*[5] with:

$$Y_{ijk} = \mu_{..} + \eta_i + \alpha_j + \beta_k + \gamma_{jk} + \epsilon_{ijk}$$

In this model, we assume that:

$\mu_{..}$ is the overall effect
η_i is the random effect of subject i
α_j is the fixed effect of factor A (*Size*) at level j
  $\alpha_j = \mu_{j.} - \mu_{..}$ with $j \in \{S, M, L\}$
β_k is the fixed effect of factor B (*Structure*) at level k
  $\beta_k = \mu_{.k} - \mu_{..}$ with $k \in \{X, D\}$
γ_{jk} is the interaction effect of factor A at level j and factor B at level k
  $\gamma_{jk} = \mu_{jk} - \mu_{j.} - \mu_{.k} + \mu_{..}$ with $k \in \{X, D\}$ and $j \in \{S, M, L\}$
ϵ_{ijk} is the random error effect

with:

$\mu_{..}$ is a constant
η_i are independent and normally distributed $N(0, \sigma_\eta^2)$
α_j are constants with $\Sigma_k \alpha_j = 0$ for all j
β_k are constants with $\Sigma_j \beta_k = 0$ for all k
γ_{jk} are constants with $\Sigma_j \gamma_{jk} = 0$ for all k and $\Sigma_k \gamma_{jk=0}$ for all j
ϵ_{ijk} are independent and normally distributed $N(0, \sigma^2)$
η_i and ϵ_{ijk} are independent
$i \in \{1, \ldots, n\}$; $j \in \{S, M, L\}$; $k \in \{X, D\}$

The properties of Y_{ijk} are the following:

the expected value $E(Y_{ijk}) = \mu_{jk} = \mu_{..} + \alpha_j + \beta_k + \gamma_{jk}$
the variance $var(Y_{ijk}) = \sigma_\eta^2 + \sigma^2$
the covariance $cov(Y_{ijk}, Y_{ij'k'}) = \sigma_\eta^2$ with not both $j = j'$ and $k = k'$

Thus, this repeated measures model assumes that the variable Y_{ijk} have constant variance, and that any two treatment observations for the same subject in advance of the random trials have constant covariance. Any two observations from different subjects in advance of the random trials are independent. Finally, all random variables are assumed to be normally distributed. Once the subjects have been selected, repeated measures model assumes that all of the treatment observations for a given subject are independent—that is, that there are no interference effects, such as order effects or carry-over effects from one treatment to the next.

4.5. Hypotheses

The initial three hypotheses to be tested in this study are the following: whether or not there is an interaction effect of the factors *Structure* and *Size* on each of the two

[5] Only the main factor effects and the interaction effect between the main factors are considered.

Table 5
Hypotheses on factor effects

Interaction effects	Null hypothesis H_0	Alternative hypothesis H_a
Structure × Size Interaction effects	H_0^1: all $\gamma_{jk} = 0$	H_a^1: not all γ_{jk} equal zero
Size Main effects	H_0^2: all $\alpha_j = 0$	H_a^2: not all α_j equal zero
Structure Main effects	H_0^3: all $\beta_k = 0$	H_a^3: not all β_k equal zero

dependent variables (*Time* and *Correctness*), and whether or not there is a main effect of each of these factors. There is *interaction* if the effect of the factor *Structure* on a dependent variable depends on the level of the factor *Size*, and vice versa. In Table 5, these null hypotheses with their alternatives are stated in terms of the model; H_0^i denotes the i-th null hypothesis, and H_a^i denotes the corresponding i-th alternative hypothesis. The level of significance $\alpha = 0.05$.

Hypotheses on nine selected pairwise comparisons of treatment means on each of the two dependent variables will be tested as well, in particular if there is interaction. The specific null hypotheses, with the alternatives, are given in Table 6.

The tests are carried out on the same data set, and therefore the tests are dependent. We will set a family level of significance of $\alpha = 0.10$. The individual significance level for each hypothesis will be derived from this value by using one of the methods for multiple comparisons [14].

5. Subjects

All subjects in the experiment are first- and second-year students at the University of Twente in Computer Science or Business Information Technology: in total 103 students participated in the experiment. They all completed successfully at least one course on Functional Programming [15].

6. Objects

The objects in the experiments are Miranda scripts with a function definition and a top expression. For each *Size* (Small, Medium and Large), an X-version of a script and a corresponding D-version is constructed. An X- or nonstructured version consists of an X-structured function definition and a top expression with an X-structured path; a D- or structured version consists of a D-structured function definition and a top expression with a D-structured path. The two paths in the two versions are similar: the X-structured path is a permutation of the D-structured path. The length of the path is for Small scripts equal to 6; for Medium scripts: 9; and for Large scripts: 12.

For each size, an X-version of a script is set up and a comparable D-version of this script. Two sets of comparable scripts are set up: e.g., script 101 is comparable to script 102, and so on (see Table 17).

Two sets are used in order to reduce practice effects (cf. Scanlan [2]). A subject tested on an X-version out of the first set will be tested on the D-version out of the second set, and vice versa (see Table 18).

An example of an X-version and a comparable D-version of small scripts (scripts 101 and 103) is given in Table 1. Two types of questions about these scripts can be distinguished: *forward questions* (for a given input to derive the possible outputs) and *backward questions* (for a given output to derive the conditions on the input) [16]. For each version an example question with the answer is given in Table 7.

Table 6
Hypotheses on treatment means for the variable time

	Treatment	Null hypothesis	Alternative hypothesis
Structure	Size at level Small	$H_o^4 : \mu_{SX} - \mu_{SD} = 0$	$H_a^4 : \mu_{SX} - \mu_{SD} \neq 0$
	Size at level Medium	$H_o^5 : \mu_{MX} - \mu_{MD} = 0$	$H_a^5 : \mu_{MX} - \mu_{MD} \neq 0$
	Size at level Large	$H_o^6 : \mu_{LX} - \mu_{LD} = 0$	$H_a^6 : \mu_{LX} - \mu_{LD} \neq 0$
Size	Structure at level X	$H_o^7 : \mu_{MX} - \mu_{SX} = 0$	$H_a^7 : \mu_{MX} - \mu_{SX} \neq 0$
		$H_o^8 : \mu_{LX} - \mu_{MX} = 0$	$H_a^8 : \mu_{LX} - \mu_{MX} \neq 0$
		$H_o^9 : \mu_{LX} - \mu_{SX} = 0$	$H_a^9 : \mu_{LX} - \mu_{SX} \neq 0$
	Structure at level D	$H_o^{10} : \mu_{MD} - \mu_{SD} = 0$	$H_a^{10} : \mu_{MD} - \mu_{SD} \neq 0$
		$H_o^{11} : \mu_{LD} - \mu_{MD} = 0$	$H_a^{11} : \mu_{LD} - \mu_{MD} \neq 0$
		$H_o^{12} : \mu_{LD} - \mu_{SD} = 0$	$H_a^{12} : \mu_{LD} - \mu_{SD} \neq 0$

Table 7
Examples of forward questions and backward questions

Question	Script 101: X-version	Script 103: D-version
Forward	The given input: [1,2,3] The conditions are: $(x : y : z : zs) \wedge \neg(x > 2) \wedge (x : xs)$ The resulting output is: 2	The given input: [3,4] The conditions are: $(x : y : z : zs) \wedge (x : xs) \wedge (x > 2)$ The resulting output is: 2
Backward	The given output: 2 The conditions on the input are: $((x : y : z : zs) \wedge (x : xs) \wedge (x > 2)) \vee$ $(\neg(x : y : z : zs) \wedge (x : xs))$	The given output: 2 The conditions on the input are: $\neg(x : y : z : zs) \wedge (x : xs) \wedge (x > 2)$

In the X-structured version (script 101) in Fig. 3 there are two paths to expression 2; in D-structured function definitions there is only one path to each expression. As can be seen in this example with forward questions, the sequence of conditions in the X-version is a permutation of the conditions in the D-version.

In this study, forward questions with simple numeric output expressions were applied to avoid problems with the skill of subjects to draw up the conditions on the input. In the experiment, the question for each script is: '*Give the value of top (1 or 2 or ... or 99 if top yields a program error)*'. The actual test objects for each subject in the experiment are six Miranda scripts each with the question about the value of *top*.

7. Procedure

The following procedure in the experiment has been established (after a pilot study with eight expert programmers) (cf. Scanlan):

- the subjects did the experiment in groups of about 20, each subject at his own PC (UNIX on PC's connected to SUN-workstations);
- the subjects answered the questions as an assignment in a regular laboratory session in the computer room;
- the subjects have been assigned randomly to one of the groups of scripts (see Table 18);
- the instruction was given on screen: there is no influence of the variability of a human instructor;
- four example scripts, with the question about the value of *top*, were offered in fixed order to each subject; the feedback on the answers is just 'correct' or 'not correct';
- after the instruction and the example questions, the six treatment scripts with questions were offered to the subjects;
- for each subject, a random permutation was used of these six scripts: this is done to balance out practice and fatigue effects; no feedback is given on the answers to these questions;
- all subjects serve in all treatments (D- and X-structured for small, medium and large scripts) resulting in a repeated measures design;
- the collection of the data on time and correctness of the answer has been automated (resulting in a log-file for each subject and a file with data of all subjects);

8. Results

In this section, the results of the experiment will be given: first, the outliers will be discussed, then the

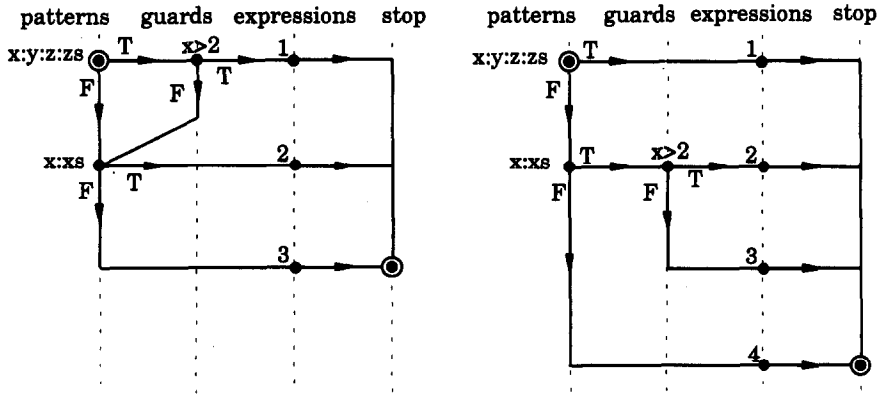

Fig. 3. Flowgraphs of functions in script 101 and 103.

Table 8
Template ANOVA table for two-factor repeated measures design with repeated measures on both factors

Source of variation	Sum of squares SS	Degrees of freedom df	Mean squares MS = SS/df
Subjects	SSS	n-1	MSS = SSS/(n-1)
Factor A	SSA	a-1	MSA = SSA/(a)
Factor B	SSB	b-1	MSB = SSB/(b-1)
Interaction AB	SSAB	(a-1)(b-1)	MSAB = SSAB/((a-1)(b-1))
Error	SSE	(n-1)(ab-1)	MSE = SSE/((n-1)(ab-1))
Total	SSTO	abn-1	

analysis of variance for *Time* and for *Correctness*. The experiment has been carried out with 103 subjects. The average time they spent on the whole experiment (instruction, example scripts, treatment scripts) was 10.5 minutes (standard deviation 2.5 minutes).

8.1. Outliers

For each subject there are six measurements on the dependent variable *Time*, i.e. for each of the treatments. For 103 subjects there are 618 time measurements. Outliers [17] on the measurement of *Time* have been detected on the basis of the externally studentized residual.[6] If for a subject the absolute value of the residual exceeds the value 3.0 then all measurements for this subject are disregarded. There appear to be 9 outliers[7] with the residual value $\geqslant 3.0$ for nine different subjects. It had been noticed that some subjects were distracted by external events during the experiment in the computer room: this could be a reason for the extreme outliers. The data of these subjects is disregarded, so of the remaining 94 subjects 564 time measurements are used in the testing of the hypothesis, together with the related measurement of the correctness.

8.2. Analysis of variance

The effects of the factors *Size* and *Structure* on the dependent variables *Time* and *Correctness* have been established. The strategy for this analysis is given by Neter et al. [14] using the variance of the data. A template of an ANOVA table [14] is given in Table 8, with a = the number of levels of factor A (*Size*: a = 3); b = the number of levels of factor B (*Structure*: b = 2); n = the sample size (the number of subjects for each treatment: n = 94). The sums of squares for each of the dependent variables have been calculated from the data[8].

In the subsequent sections, the influence of the factors *Size* and *Structure* will be analysed for each of the dependent variables *Time* and *Correctness*:

(1) A summary statistic will be given with treatment means and factor level means variable (with sample standard deviations) according to Table 4.
(2) The treatment means for structured and nonstructured function definitions will be displayed as function of the *Size*.
(3) An analysis of variance will be presented according to Table 8.
(4) The hypotheses on the interaction effects and the factor effects will be tested.

8.3. Time

For each treatment, the sample mean *Time* in seconds (and the standard deviation) is given in Table 9, together with the factor level means. The sample treatment mean $m_{jk} = \Sigma_i y_{ijk}/n$, where y_{ijk} is the observed value on the dependent variable Y_{ijk}.

In Fig. 4 the sample treatment means of the variable *Time* for structured (D) and nonstructured (X) function definitions are displayed as functions of the *Size*.

In case of this continuous dependent variable, we can thus use the F*-statistic which has the F-distribution under the null hypothesis. The decision rules are as follows:

- Interaction effect AB (Size × Structure) F* = MSAB/MSE.

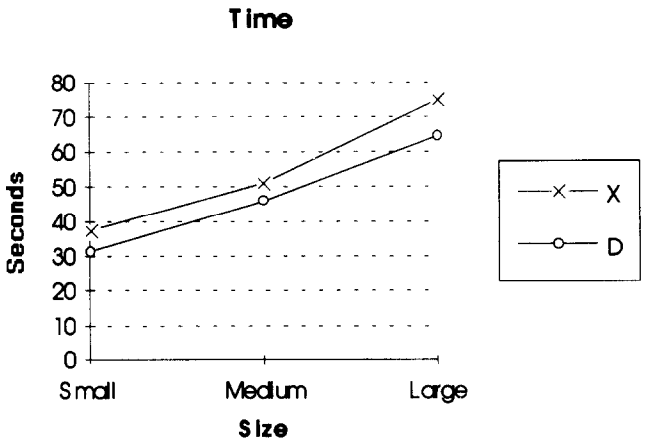

Fig. 4. The means of *Time* (seconds) measured for X- and D-structured scripts as function of the *Size*.

[6] In SPSS called the studentized deleted residual.
[7] Number of outliers on *Time* per treatment: SX 1; SD 2; MX 1; MD 3; LX 0; LD 2.
[8] With SPSS for Windows.

Table 9
The sample mean Time (seconds) and the standard deviation (n = 94)

Factor		Size			
	Levels	Small	Medium	Large	Level mean
Structure	Nonstructured X	37.40	50.89	74.96	54.42
		(18.81)	(24.95)	(35.97)	
	Structured D	31.46	45.80	64.49	47.25
		(16.97)	(18.94)	(24.89)	
	Level Mean	34.43	48.35	69.73	50.83

If $F^* \leqslant F[1\text{-}\alpha; (a\text{-}1)(b\text{-}1), (n\text{-}1)(ab\text{-}1)]$ we fail to reject the null hypothesis H_0^1; otherwise the null hypothesis is rejected and we accept the alternative hypothesis H_a^1.

- Main effect factor A (Size) with $F^* = MSA/MSE$. If $F^* \leqslant F[1\text{-}\alpha; (a\text{-}1), (n\text{-}1)(ab\text{-}1)]$ we fail to reject the null hypothesis H_0^2; otherwise the null hypothesis is rejected and we accept the alternative hypothesis H_a^2.
- Main effect factor B (Structure) with $F^* = MSB/MSE$. If $F^* \leqslant F[1\text{-}\alpha; (b\text{-}1), (n\text{-}1)(ab\text{-}1)]$ we fail to reject the null hypothesis H_0^3; otherwise the null hypothesis is rejected and we accept the alternative hypothesis H_a^3.

The ANOVA table for the dependent variable *Time* is given in Table 10, together with the calculated F^*-value, the F-value at significance level α, and also the p-value (the probability, when H_0 is true, of observing a test result as deviant or more deviant than the result actually obtained).

From Table 10, with a level of significance $\alpha = 0.05$, it can be concluded that:

- There is no significant interaction effect between *Size* and *Structure* on the variable *Time*, since $F^* \leqslant F$ (p = .443) (the curves of the treatment means in Fig. 4 for the two levels of *Structure* are nearly parallel): i.e., we fail to reject the null hypothesis H_0^1.
- There is a significant main effect of *Size* on the variable *Time*, since $F^* > F$ (p = .000). This means that the null hypothesis H_0^2 can be rejected.
- There is a significant main effect of *Structure* on the variable *Time*, since $F^* > F$ (p = .000). This means that the null hypothesis H_0^3 can be rejected.

The hypotheses $H_0^4, .., H_0^{12}$, involving the treatment means, can be tested as well. The family level of significance is chosen to be $\alpha = 0.10$. There are 9 pairwise comparisons of treatment means, each of them can be analysed with a single degree of freedom test [14] with $\alpha' = 0.10/9 = 0.011$. The t^* test statistic has been used, with $t^* = (m_{jk}\text{-}m_{j'k'})/\sqrt{(2 \times MSE/n)}$, and degrees of freedom = (n-1)(ab-1). Under the null hypothesis, the statistic t^* follows the t distribution. Here, $t[0.11; 465] = 2.33$.

From Table 11, it can be concluded that in tests 4 and 5 the null hypothesis cannot be rejected; in the other tests (6..12) the null hypothesis can be rejected and the corresponding alternative hypothesis will be accepted. In other words, in these cases there is a significant influence (with a family level of significance $\alpha = 0.10$) on the dependent variable *Time*.

If there had been an interaction effect, the effect of *Size* on the *Structure*-effect could have been tested with the following hypotheses:

$$H_0^{13} : (\mu_{SX}-\mu_{SD}) - (\mu_{MX}-\mu_{MD}) = 0$$

$$H_0^{14} : (\mu_{MX}-\mu_{MD}) - (\mu_{LX}-\mu_{LD}) = 0$$

$$H_0^{15} : (\mu_{SX}-\mu_{SD}) - (\mu_{LX}-\mu_{LD}) = 0$$

These tests could replace the ratio-measure as defined by Scanlan [2], as will be argued in the discussion section.

8.4. Correctness

For each treatment, the sample mean of the variable *Correctness* (and the standard deviation) is given in Table 12, together with the factor level means. The

Table 10
ANOVA table for the variable *Time*

Source of variation	Sum of squares	df	Mean squares	F^*	F $\alpha = 0.05$	p
Subjects	96382	93	1036.37			
Factor *Size*	118828.1	2	59414.05	117.6	3.00	.000
Factor *Structure*	7249.09	1	7249.09	13.68	3.84	.000
Size × *Structure* interactions	783.78	2	391.89	0.820	3.00	.443
Error	232469.4	465	499.93			
Total	455712.3	563				

Table 11
Single degree of freedom tests for hypotheses on the variable *Time*

Null hypothesis	Estimated	t^*	$t^* \geqslant t$	p
$H_o^4 : \mu_{SX} - \mu_{SD} = 0$	$m_{SX} - m_{SD} = 5.94$	1.82	False	.036
$H_o^5 : \mu_{MX} - \mu_{MS} = 0$	$m_{MX} - m_{MD} = 5.09$	1.56	False	.070
$H_o^6 : \mu_{LX} - \mu_{LD} = 0$	$m_{LX} - m_{LD} = 10.47$	3.21	True	.001
$H_o^7 : \mu_{MX} - \mu_{SX} = 0$	$m_{MX} - m_{SX} = 13.49$	4.14	True	.000
$H_o^8 : \mu_{LX} - \mu_{MX} = 0$	$m_{LX} - m_{MX} = 24.07$	7.38	True	.000
$H_o^9 : \mu_{LX} - \mu_{SX} = 0$	$m_{LX} - m_{SX} = 37.56$	11.52	True	.000
$H_o^{10} : \mu_{MD} - \mu_{SD} = 0$	$m_{MD} - m_{SD} = 14.34$	4.40	True	.000
$H^{11} : \mu_{LD} - \mu_{MD} = 0$	$m_{LD} - m_{MD} = 18.69$	5.73	True	.000
$H_o^{12} : \mu_{LD} - \mu_{SD} = 0$	$m_{LD} - m_{SD} = 33.03$	10.13	True	.000

Table 12
Proportion correct answers and sample standard deviation (n = 94)

Factor		Size			
	Levels	Small	Medium	Large	Level mean
Structure	Nonstructured X	0.660	0.600	0.670	0.643
		(0.48)	(0.49)	(0.47)	
	Structured D	0.710	0.860	0.940	0.836
		(0.45)	(0.35)	(0.25)	
	Level mean	0.685	0.730	0.805	0.740

sample mean gives the *proportion* correct answers; this can also be seen as the probability of a correct answer.

In Fig. 5 the proportion correct answers for structured (D) and nonstructured (X) function definitions are displayed as function of the *Size*. In case of this binary dependent variable, we will use the Q-statistic, defined by Cochran [17], with a χ^2-distribution under the null hypothesis; $\chi^2[1\text{-}\alpha, df]$ denotes the χ^2-value at significance level α and df is the degrees of freedom. The decision rules are as follows:

- Interaction effect AB (Size × Structure) with Q = SSAB/MSE.

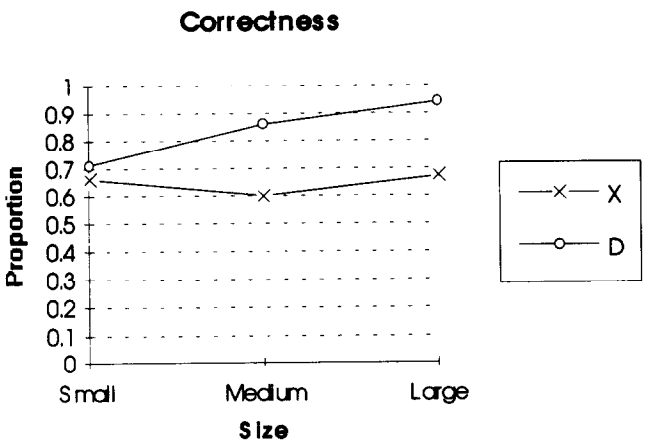

Fig. 5. The proportion *Correct* answers for X- and D-structured scripts as function of the *Size*.

If $Q \leqslant \chi^2[1\text{-}\alpha; (a\text{-}1)(b\text{-}1)]$ we fail to reject the null hypothesis H_0^1; otherwise the null hypothesis is rejected and we accept the alternative hypothesis H_a^1.

- Main effect factor A (Size) with Q = SSA/MSE. If $Q \leqslant \chi^2[1\text{-}\alpha; (a\text{-}1)]$ we fail to reject the null hypothesis H_0^2; otherwise the null hypothesis is rejected and we accept the alternative hypotheses H_a^2.
- Main effect factor B (Structure) with Q = SSB/MSE. If $Q \leqslant \chi^2[1\text{-}\alpha; (b\text{-}1)]$ we fail to reject the null hypothesis H_0^3; otherwise the null hypothesis is rejected and we accept the alternative hypothesis H_a^3.

The ANOVA table for the dependent variable *Correctness* is given in Table 13, together with the calculated value for the Q-statistic, the χ^2-value at significance level α, and the p-value.

From Table 13, with a level of significance $\alpha = 0.05$, it can be concluded that:

- There is a significant interaction effect between *Size* and *Structure* on the variable *Correctness*, since $Q > \chi^2$ (p = 0.021). This means that the null hypothesis H_0^1 can be rejected.
- There is a significant main effect of *Size* on the variable *Correctness*, since $Q > \chi^2$ (p = .000). This means that the null hypothesis H_0^2 can be rejected.
- There is a significant main effect of *Structure* on the variable *Correctness*, since $Q > \chi^2$ (p = 0.016). This means that the null hypothesis H_0^3 can be rejected.

Again, we will consider the hypotheses on the

Table 13
ANOVA table for the variable *Correctness*

Source of variation	Sum of squares	df	Mean squares	Q	χ^2 $\alpha = 0.05$	p
Subjects	30.19	93	0.32			
Factor *Size*	1.32	2	0.66	7.90	5.99	.021
Factor *Structure*	5.36	1	5.36	32.09	3.84	.000
Size × *Structure* interactions	1.42	2	0.71	9.50	5.99	.016
Error	70.39	465	0.15			
Total	108.68	563				

Table 14
Single degree of freedom tests for hypotheses on the variable *Correctness*

Null hypothesis	Estimated	M	$M \geqslant \chi^2$.	p
$H_0^4 : \mu_{SX} - \mu_{SD} = 0$	$m_{SX} - m_{SD} = -0.05$	0.86	False	.353
$H_0^5 : \mu_{MX} - \mu_{MD} = 0$	$m_{MX} - m_{MD} = -0.26$	16.89	True	.000
$H_0^6 : \mu_{LX} - \mu_{LD} = 0$	$m_{LX} - m_{LD} = -0.27$	17.86	True	.000
$H_0^7 : \mu_{MX} - \mu_{SX} = 0$	$m_{MX} - m_{SX} = -0.06$	1.06	False	.304
$H_0^8 : \mu_{LX} - \mu_{MX} = 0$	$m_{LX} - m_{MX} = 0.07$	1.32	False	.250
$H_0^9 : \mu_{LX} - \mu_{SX} = 0$	$m_{LX} - m_{SX} = 0.01$	0.03	False	.853
$H_0^{10} : \mu_{MD} - \mu_{SD} = 0$	$m_{MD} - m_{SD} = 0.15$	8.17	True	.004
$H_0^{11} : \mu_{LD} - \mu_{MD} = 0$	$m_{LD} - m_{MD} = 0.08$	3.27	False	.071
$H_0^{12} : \mu_{LD} - \mu_{SD} = 0$	$m_{LD} - m_{SD} = 0.23$	16.33	True	.000

treatment means. The family level of significance is chosen to be $\alpha = 0.10$. There are nine pairwise comparisons of treatment means, each of them can be analysed with a single degree of freedom test with $\alpha' = 0.011$. For each comparison, we can use the McNemar-statistic M with a χ^2-distribution under the null hypothesis. M is estimated as follows [18]: $M = (n_{01} - n_{10})^2/(n_{01} + n_{10})$, where n_{xy} is the number of observations having response x on the first treatment in the comparison, and response y on the second treatment (response 0 = incorrect; response 1 = correct). Furthermore, $\chi^2[1\text{-}\alpha; df] = \chi^2[0.989; 1] = 6.63$.

From Table 14, it can be concluded that in tests 4, 7, 8, 9 and 11 the null hypothesis cannot be rejected. In the other tests 5, 6, 10 and 12 the null hypothesis can be rejected: there is a significant influence (with a family level of significance $\alpha = 0.10$) on the dependent variable *Correctness*.

9. Discussion

In the following tables, the results from the previous sections have been summarized. The existence of significant main effects and interaction effects, based on the overall analysis of variance ($\alpha = 0.05$), are given in Table 15.

For each of the dependent variables *Time* and *Correctness*, there is an overall significant influence of the factor *Structure* and *Size*. For the variable *Time*, no significant interaction effect has been found; for *Correctness*, a significant interaction effect has been shown.

The results of testing the hypotheses involving treatment means are given in Table 16: whether or not the null hypothesis has been rejected (family level of significance $\alpha = 0.10$).

With these results, it can be seen that:

- The overall significant effect of *Structure* on the variable *Time* appears to be mainly due to the effect of the scripts of size *Large*.
- The overall significant effect of *Structure* on *Correctness* appears to be mainly due to the effect of the Medium and Large scripts.
- The overall significant effect of *Size* on X-structured and D-structured scripts on the variable *Time* is confirmed on each comparison.
- The overall significant effect of *Size* on the variable *Correctness* is mainly due to the effect of *Size* on D-structured scripts. For none of the comparisons on X-structured scripts, a significant influence of the factor

Table 15
Existence of significant factor effects and interaction effects on the variables *Time* and *Correctness*

Factor	Structure	Size	Structure × Size
Time	Yes	Yes	No
Correctness	Yes	Yes	Yes

Table 16
Rejection of null hypotheses 4..12 on treatment means

H_0	H_0^4	H_0^5	H_0^6	H_0^7	H_0^8	H_0^9	H_0^{10}	H_0^{11}	H_0^{12}
Time	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Correct	No	Yes	Yes	No	No	No	Yes	No	Yes

Size has been shown. This also shows the interaction between the factors *Size* and *Structure* on the variable *Correctness*.

The two dependent variables—*Time* and *Correctness*—have been taken as criteria for the performance of programmers. We assumed that the performance on structured function definitions versus comparable nonstructured function definitions corresponds to the performance on D-structured paths versus the similar X-structured paths as tested in the hypotheses. Then, in summary, we conclude that, with respect to the structure and size of function definitions:

(1) Subjects need significant *less time* to obtain an answer to structured function definitions than to nonstructured function definitions.
(2) Subjects give significant *more often correct* answers to somewhat larger structured function definitions than to nonstructured function definitions of comparable size.
(3) Subjects need significant *more time* to obtain an answer to larger function definitions than to smaller ones.
(4) Subjects give significant *more often correct* answers for larger structured function definitions than for smaller ones.

Conclusions 1 and 2 give empirical evidence to support the general conclusion that programmers perform better on structured Miranda function definitions than on nonstructured definitions.

Conclusion (3) seems to be quite obvious, but conclusion (4) came up as rather a surprise and seems to be counter-intuitive: an increase in the proportion correct answer for larger function definitions. However, a similar 'unexpected' trend has been observed in other studies: Basili and Perricone [19] found that there is a higher error rate in smaller sized modules than in larger modules. One of the tentative explanations they offer is that larger modules are coded with more care than smaller modules because of their size. Also Möller and Paulish [20] found significantly higher fault rate in small modules as compared to larger ones.

The experiment used in this study is similar to the one used by Scanlan [2] in the comparison of flowcharts and pseudocode. However, the design and statistical analysis used in this study differ on some important aspects.

- Scanlan used a single factor repeated measures design, as opposed to a two-factor repeated measures design used here. In our study, the main effects and interaction effect have been established on the basis of analysis of the variance; the dependency of hypotheses on the treatment means has been accounted for explicitly.
- A ratio-measure is used by Scanlan in order to assess the interaction effect. In terms of the present study, the ratio is calculated by dividing the larger time (of the structured or nonstructured definition) by the smaller time (of the structured or nonstructured definition) for each subject at each size level; those ratios in favour of structured definitions receive a positive sign; those ratios in favour of nonstructured definitions receive a negative sign. In our experiment, the ratio-measure resulted in a highly non-normal distribution, because of the discontinuity of the measure between -1 and $+1$. Furthermore, in case of equal times, there is no appropriate decision rule to assign the value -1 or $+1$. In this paper, an alternative is proposed for the ratio-measure.
- The confidence measure used by Scanlan is on an ordinal scale: four levels from 1 to 4. The mean confidence level of such an ordinal measure, as used by Scanlan, is questionable. Moreover, it is not obvious that the subjects are reliable in the self-assessment of the correctness of their solution, in other words whether the confidence level depends on the correctness of their answer. In some other studies it has been shown that this is not always the case. Gibson and Senn [21] found a notable discrepancy between correctness and confidence. Gathy and Denef [22] found a strong correlation between the self-confidence assessment scores and the final examinations for good students, whereas a negative but loose correlation was observed for weak students. Leclercq [23] analysed factors that affect the confidence estimation and the confidence expression.

With respect to these points, Scanlan's study [2] should be reconsidered.

Table 17
Two sets of scripts

Size		Small	Medium	Large
Set 1	X-version	101	105	109
	D-version	102	106	110
Set 2	X-version	104	108	112
	D-version	103	107	111

Table 18
Scripts (X- and D-versions) for two groups of subjects

	Small		Medium		Large	
Group 1	101 X	103 D	105 X	107 D	109 X	111 D
Group 2	102 D	104 X	106 D	108 X	110 D	112 X

10. Conclusion

The aim of this study has been to investigate programmers' performance on structured versus nonstructured function definitions. In the experiment, based on a two-factor repeated measures design, the control-flow model of Miranda function definitions and related metrics proved to be useful in the definition the factors and factor levels of *Structure* and *Size*.

The experimental findings support the main hypothesis that programmers perform better on structured Miranda function definitions than on nonstructured definitions. Some counter-intuitive findings, reported in the literature before, came up in the present study as well: programmers make fewer errors in larger function definitions than in smaller ones.

Based on these experimental findings, the programming style rule can be put forward to use structured function definitions instead of nonstructured ones. This would mean that a programming style is adopted to write guards that always are concluded with an 'otherwise'-case. The rule could be relaxed by demanding *total* guards, such that always, once a pattern succeeds, one of the guards in the clause will succeed. In that situation, if no 'otherwise'-case is used to obtain total guards, a nonstructured function definition would be obtained in the control-flow model, because in the model is abstracted from the actual content of the guards.

To check the application of this programming rule, a Miranda static analyser [3] based on the control-flow model of function definitions can be used. With this analyser, X-structured function definitions in scripts can be spotted easily, also in scripts with many definitions. After this anomaly checking, these definitions can be inspected on errors and/or be rewritten to a structured version.

In a survey of scripts written by experts, hardly any nonstructured function definition has been found. Apparently, experts already do not use this kind of function definition. For some programmers with a few years of functional programming experience, nonstructured function definitions have been detected in their scripts. Programming style rules, as proposed above, could make programmers aware of the operational semantics of function definitions.

Acknowledgements

The authors would like to thank N. Fenton for his comments on an earlier version of this paper; R. Houterman and S. Oosterloo for their advice on the statistics used in this paper; E. Prangsma and M. Harssema for the assistance in the experiment; and last but not least, the students for their participation in the experiment.

Appendix

Versions of Miranda scripts (Table 17) and the allocation to subjects in experiment (Table 18).

References

[1] I. Vessey and R. Weber, Research on structured programming: an empirist's evaluation, IEEE Trans. SE, 10 (1984) 397–407.
[2] D.A. Scanlan, Structured flowcharts outperform pseudocode: an experimental comparison, IEEE Software (September 1989) 28–36.
[3] K.G. van den Berg and P.M. van den Broek, Static analysis of functional programs, Inf. and Soft. Technol., 37 (1995) 213–224.
[4] V.R. Basili, R.W. Selby and D.H. Hutchens, Experimentation in Software Engineering, IEEE Trans. SE, 12 (1986) 733–743.
[5] D.A. Turner, An overview of Miranda, Sigplan Notices, 21 (1986) 158–166.
[6] R. Bird and P. Wadler, Introduction to Functional Programming, Prentice-Hall, New York, 1988.
[7] R.S. Pressman, Software Engineering, A Practitioner's Approach, McGraw-Hill, New York, 3rd edn. 1992.
[8] S.L. Peyton Jones, The Implementation of Functional Programming Languages, Prentice-Hall, New Jersey, 1987.
[9] I. Holyer, Functional Programming with Miranda, Pitman, London, 1991.
[10] M. Petre and R. Winder, On languages, models and programming styles, The Computer J., 33 (1990) 173–180.
[11] R. Plasmeijer and M. van Eekelen, Functional Programming and Parallel Graph Rewriting, Addison-Wesley, Wokingham, 1993.
[12] N.E. Fenton, Software Metrics: A Rigorous Approach, Chapman & Hall, London, 1991.
[13] T. Moher and G.M. Schneider, Methodology and experimental research in software engineering, Int. J. Man-Machine Studies, 16 (1982) 65–87.
[14] J. Neter, W. Wasserman and M.H. Kutner, Applied Linear Statistical Models. Regression, Analysis of Variance, and Experimental Designs, 3rd edn., Irwin, Homewood, 1990.
[15] S.M.M. Joosten (ed.), K.G. van den Berg and G.F. van der Hoeven, Teaching functional programming to first-year students, Journal of Functional Programming, 3 (1993) 49–65.

[16] T.R.G. Green, IF's and THEN's: is nesting just for the birds?, Software-Practice and Experience, 10 (1980) 373–381.
[17] J.L. Myers and A.D. Well, Research Design and Statistical Analysis, HarperCollins, New York, 1991.
[18] S. Kotz and N.L. Johnson (eds.), Encyclopedia of Statistical Sciences, Wiley, New York, 1989.
[19] V.R. Basili and B.T. Perricone, Software errors and complexity: an empirical investigation, Comm. ACM, 27 (1984) 42–52.
[20] K.-H. Möller and D.J. Paulish, An empirical investigation of software fault distribution, 1st Int. Software Metrics Symposium Baltimore, IEEE, Washington, 1993, pp. 82–90.
[21] V.R. Gibson and J.A. Senn, System structure and software maintenance performance, Comm. ACM, 32 (1989) pp. 347–358.
[22] P. Gathy and J.-F. Denef, Self-assessment during computer-assisted testing in histology, in D.A. Leclercq and J.E. Bruno (eds.), Item Banking: Interactive Testing and Self-Assessment, Springer, Berlin, 1993, pp. 233–241.
[23] D.A. Leclercq, Validity, reliability, and acuity of self-assessment in educational testing, in D.A. Leclercq and J.E. Bruno (eds.), Item Banking: Interactive Testing and Self-Assessment, Springer, Berlin, 1993, pp. 114–131.