

A Controlled Experiment To Assess the Benefits of Procedure Argument Type Checking

But First: Validity

- **Internal validity:** the extent to which a piece of evidence supports a claim about cause and effect, within the context of a particular study.
- Example threats:
 - Confounding (e.g., observational study: more-skilled programmers choose Haskell)
 - Selection bias (e.g., best programmers end up in one group)
 - Testing effects (e.g., Hawthorne effect)
 - Shortened the day by 30 minutes (output went up); shortened it more (output per hour went up, but overall output decreased); returned to the first condition (output peaked).

External Validity

- External validity is the validity of applying the conclusions of a scientific study outside the context of that study.
- Threats to external validity?

Construct Validity

- Are you measuring what you intend to measure?
- e.g., goal: assess software engineer quality as a function of various variables
- Attempt: measure lines of code added per year (Musk asked Twitter employees to print out their last 30 days of code??)
- Attempt: measure bugs fixed per year

ANSI Vs. K&R C

K&R C:

```
int foo(s, f, b)
    char* s;
    float f;
    struct Baz * b;
{
    return 5;
}

foo(3, 4, 5); // OK
```

ANSI C:

```
int foo(char* s,
        float f,
        struct Baz * b)
{
    return 5;
}

foo(3, 4, 5); // ERROR: 3 is not a char *...
```


Hypotheses

- Hypothesis 1. Type checking increases Interface Use Productivity.
- Hypothesis 2. Type checking reduces the number of Interface Defects in delivered programs.
- Hypothesis 3. Type checking reduces Interface Defect Lifetimes.

Participants

- The 34 subjects had the following education. Two were postdoctorates 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 subjects had between 4 and 19 years of programming experience ($\mu = 10$) and all but 11 of them had written at least 3,000 lines in C (all but one at least 300 lines). Only eight of the subjects had some programming experience with X-Windows or Motif; only three of them had written more than 300 lines in X-Windows or Motif.

Tasks

- Defined at a low level of granularity

```
/* 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);
```


Task Evaluation

- Hand-assessed

ANSI C Makes the Second Task Faster

TABLE 2
OVERALL PRODUCTIVITY STATISTICS

Statistic		both tasks		1st task		2nd task	
		ANSI	K&R	ANSI	K&R	ANSI	K&R
1	hours to delivery	1.3	1.35	1.6	1.6	0.9	1.3
	$p =$	0.49		0.83		0.018	
2	#versions	15	16	19	21	12.5	13
	$p =$	0.84		0.63		0.16	
3	FU/hr	8.6	9.7	7.2	8.5	12.8	10.7
	$p =$	0.93		0.31		0.061	

Medians 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 first + second, first, and second problem tackled chronologically by each subject, respectively. All entries include data points for both problem A and problem B.

ANSI C Helps Remove Defects Faster

TABLE 3
STATISTICS ON INTERNALS OF THE PROGRAMMING PROCESS

Statistic		both tasks		1st task		2nd task	
		ANSI	K&R	ANSI	K&R	ANSI	K&R
4	accumul. interface detect lifetime (median)	0.3	1.2	0.5	2.1	0.2	1.1
	<i>p</i> =	0.004		0.028		<i>0.059</i>	
5	#right, then wrong again (75% quantile)	1.0	1.0	1.0	1.0	0.0	1.0
	<i>p</i> =	<i>0.12</i>		<i>0.82</i>		0.009	

See Table 2 for explanations.

ANSI C Results in Better Programs

TABLE 4
STATISTICS ON THE DELIVERED PROGRAM

Statistic		both tasks		1st task		2nd task	
		ANSI	K&R	ANSI	K&R	ANSI	K&R
6	#gaps (75% quantile)	0.25	0.0	1.5	0.0	0.0	0.0
	$p =$	<i>0.35</i>		<i>0.26</i>		<i>0.70</i>	
7	#errors remaining in delivered program	1.0	2.0	1.0	2.0	1.0	2.0
	$p =$	0.0016		<i>0.32</i>		0.031	
8	–for <i>invisD</i> only (90% quantile)	1.0	1.0	0.0	1.4	0.0	0.0
	$p =$	0.04		0.048		<i>0.41</i>	
9	–for <i>severe</i> only	1.0	1.0	1.0	0.0	1.0	1.0
	$p =$	<i>0.66</i>		<i>0.74</i>		<i>0.65</i>	
10	–for <i>severeD</i> only	0.0	1.0	0.0	1.0	0.0	1.0
	$p =$	0.0001		0.015		0.0022	

See Table 2 for explanations. Lines 6 and 8 do not list medians but other quantiles instead, as indicated.

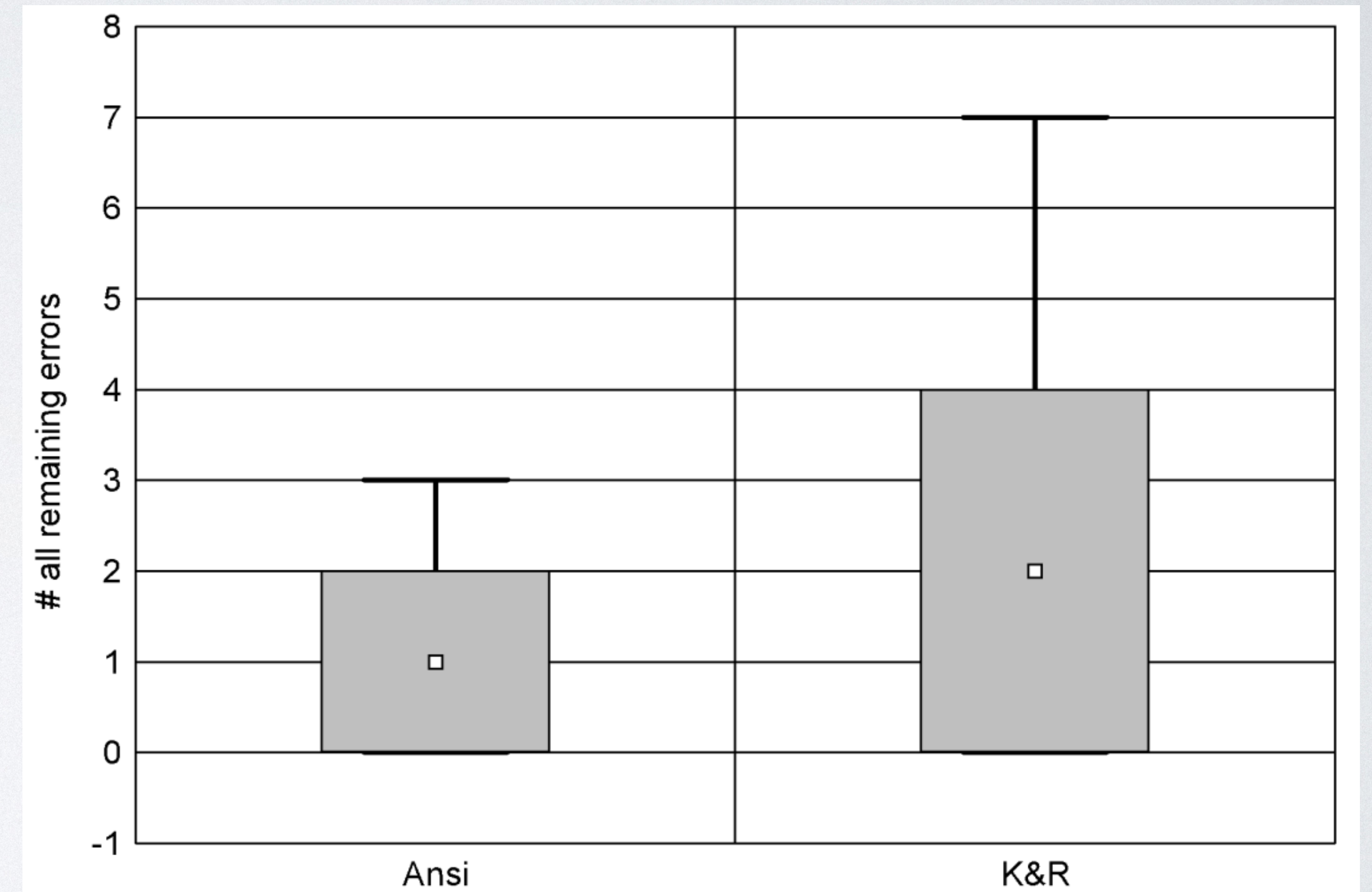


Fig. 7. Boxplots of total number of remaining defects in delivered programs over both tasks.