Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

The CERT Function Extraction Experiment: Quantifying FX Impact on Software Comprehension and Verification

Rosann W. Collins
University of South Florida and
CERT, Software Engineering Institute

Gwendolyn H. Walton
CERT, Software Engineering Institute

Alan R. Hevner
University of South Florida and
CERT, Software Engineering Institute

Richard C. Linger
CERT, Software Engineering Institute

CMU/SEI-2005-TN-047

December 2005

Survivable Systems Engineering

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

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Table of Contents

List of Tables

Executive Summary

Function Extraction (FX) is an emerging technology that can be applied to automated calculation of the functional behavior of software for improved human understanding and analysis [Pleszkoch 04, Hevner 05]. To better understand the impact of FX on software comprehension and verification, a rigorous, controlled experiment was performed to compare traditional manual methods of comprehension with automated behavior computation using an FX prototype. The experiment required 26 experienced Java programmers (13 using traditional techniques and 13 using FX automation) to evaluate the behavior of three small programs of low-to-moderate complexity. The following observations summarize the experimental results:

- **Use of the FX prototype greatly reduces the time required to derive program behavior by automating this crucial and time-consuming part of program comprehension**. Subjects using traditional manual methods spent significantly more time (between 62% and 81% of total task time) reading and analyzing code to determine program behavior, and this percentage increased as the programs became longer and more difficult. In contrast, subjects using FX automation determined program behavior directly from the prototype output and thus spent very little time (between .2% and .3% of total task time) on program comprehension. *This represents an improvement of several orders of magnitude.*
- **Use of the FX prototype improves human performance in program comprehension**. The subjects who used the FX prototype produced significantly more correct answers to comprehension and verification questions than the subjects using manual techniques; in the case of the longest, most difficult program correct answers increased by a factor of 3.6. The FX group also required significantly less time to achieve this improved comprehension. This difference grew as the programs increased in length and difficulty, ultimately resulting in a reduction of time required by a factor of 4.2. Treating productivity as a ratio of task output to input (output = accurate program comprehension, input = total time on task), *the FX group achieved a productivity improvement on the order of a factor of 15 for the longest and most difficult program.*
- **Developers who were trained on and used the FX prototype agree that it was useful, supports the comprehension task, and is easy to use**. The large improvement in program comprehension and productivity for the group using the FX prototype was achieved with just *45 minutes of instruction, contrasted with years of training and experience* in program reading and inspection for the group using traditional methods.

Standard statistical tests of the significance of the experimental data indicate extremely low probabilities that these results could be attributed to chance, in some cases computed as zero to three decimal places.

Abstract

Function Extraction (FX) is a new, theory-based technology for automated calculation of the functional behavior of software. The CERT Function Extraction experiment was conducted so as to better understand the impact of FX on human comprehension and verification of software and to rigorously quantify the business case for FX technology. This report describes the results of the controlled experiment that was performed to compare traditional manual methods of comprehension with automated behavior computation using an FX prototype. The results of the experiment show a substantial increase in human capabilities for software comprehension and verification using FX technology.

1 Function Extraction Research Motivation

Because of the size and complexity of programs, current-generation software engineering must operate in a world of incomplete knowledge of program behavior. No practical means exist for programmers to determine the full functional behavior of sizable programs in all circumstances of use, and no testing effort, no matter how extensive, can exercise more than a small fraction of possible behavior. Lacking better technology, behavior discovery today is a haphazard and time-consuming drain on resources carried out by manual techniques of program reading and inspection with unavoidable human fallibility. Yet comprehensive knowledge of software behavior is essential for fast and correct development, testing, maintenance, and evolution of programs.

While this problem is pervasive today, it need not be so in the future. A key enabling capability for next-generation software engineering is the transformation of program behavior analysis from an error-prone, resource-intensive process in human-time scale into a precise, automated calculation in CPU-time scale. The emerging technology of Function Extraction holds promise towards making this capability a reality.

2 Concepts of Function Extraction

Function Extraction (FX) deals with the semantics of software behavior. All levels of abstraction in the development of software systems embody behavioral semantics, from low-level machine language operations to high-level system capabilities. As software systems are developed and evolve over time, semantic content is continuously created, intentionally or unintentionally, correct or incorrect. Effective development and evolution of a system depends on how well its behavioral semantics are understood. The complexity and quantity of accumulated behavioral semantics can overwhelm developers, leading to loss of intellectual control.

The ultimate goal of Function Extraction is to calculate full semantic behavior at all levels of system abstraction, from specification to design to implementation. This goal requires automating the computation and composition of behaviors in the languages employed to express such artifacts. These languages, whatever their level of abstraction, embody definitions of the behavioral semantics of their structures. Function Extractor development begins with a well-defined language whose semantics can be captured in terms of the functions of its structures and the rules that govern their combination. Any system artifact written in that language can then be submitted to the Function Extractor, which will apply the functional semantics of the structures to produce as output a catalog containing all cases of behavior defined by the artifact. This behavior is expressed in non-procedural form, essentially defining the as-built specification of the artifact in terms of its mapping of inputs into outputs.

In a miniature illustration, consider the following sequence of operations on small integers x and y (machine precision is set aside; however, the semantics of finite operations could be incorporated if necessary):

```
do
  x := x + y
  y := x - y
  x := x - y
enddo
```

In this case, the behavioral semantics of each operation involves deriving the value of the right-hand-side expression and assigning it to the variable on the left. The rule of combination for a sequence of operations is ordinary function composition, easily expressed in the following trace table and associated algebraic derivations that compute the net functional behavior from input to output in non-procedural terms:

Assignment	Value of x	Value of y
x := x + y	x1 = x0 + y0	y1 = y0
y := x - y	x2 = x1	y2 = x1 - y1
x := x - y	x3 = x2 - y2	y3 = y2

```
x3 = x2 - y2            y3 = y2
   = x1 - (x1 - y1)        = x1 - y1
   = y1                    = x0 + y0 - y0
   = y0                    = x0
```

Thus, the computed behavior is

x, y := y, x

that is, the values of x and y are exchanged by the sequence of operations. It is important to note that this computed behavior represents exactly what the program does; it is now unnecessary to read and inspect the code in an attempt to derive this information. This example illustrates the Function Extraction process for a sequence control structure. A function theorem defines the mapping of all control structures into such functional forms, and is the mathematical foundation of FX technology.

In a more general explanation, the function-theoretic model of software treats programs as rules for mathematical functions or relations [Hevner 02, Hevner 05, Hoffman 01, Linger 79, McCarthy 63, Mills 86, Mills 02, Pleszkoch 90, Pleszkoch 04, Prowell 99]. While sizable programs can contain a virtually infinite number of execution paths, they are constructed of a finite number of nested and sequenced control structures, each of which makes a finite contribution to overall behavior. These structures correspond to mathematical functions or relations, that is, mappings from inputs to outputs. These functional mappings can be automatically extracted in a stepwise process that traverses the finite control structure hierarchy. At each step, details of local code and data are abstracted out, while their net effects are preserved and propagated in the extracted behavior. While no general theory for loop abstraction can exist, use of recursive expressions and patterns for loops provides an engineering solution.

Function Extraction has potential for widespread application across the software engineering life cycle, as discussed in Hevner [Hevner 05]. Of special interest is use of FX for maintaining and evolving legacy systems, as well as for developing new systems. The technology can also play a key role in understanding malicious code; CERT is currently developing an FX-based system for analyzing malicious code expressed in Intel Assembler Language. And because behavior computation is a key part of both software verification at the program level and component composition at the system level, FX application is possible in these areas as well.

3 The Function Extraction Controlled Experiment

The CERT® organization of the Carnegie Mellon® Software Engineering Institute has implemented Function Extraction technology in an FX prototype that operates on a small subset of the Java programming language. The prototype takes in a Java program written in the language subset and automatically calculates and displays its functional behavior. The behavior is expressed in a non-procedural, user-readable format in terms of how the outputs of the program are produced from its inputs in all possible uses, in effect producing the as-coded specification of the program.

CERT has performed a formal, controlled experiment to quantify the impact of FX technology on the ability of programmers to comprehend and verify programs. The experimental subjects were 26 Carnegie Mellon University (CMU) graduate students with substantial computer science education and software development experience. The experiment was approved by the CMU Institutional Review Board (IRB), calibrated in two pilot tests, and conducted according to rigorous experimental protocols.

The 26 subjects signed IRB consent forms and received 45 minutes of classroom instruction on the purpose of the experiment and the process of program comprehension. They were then randomly divided into two groups: the control group (manual manipulation) and the experimental group (automated FX manipulation):

- The control group subjects were given three Java programs of varying size and difficulty, together with functional requirements for the programs and a set of questions to answer. They applied traditional manual methods of reading and inspection to understand the behavior of each program and then answered the questions. All activities were self-timed by the subjects, and a post-hoc questionnaire was completed.
- The experimental group subjects installed the Function Extraction prototype on their personal laptop computers. These subjects, who had no previous exposure to the prototype, received an additional 45 minutes of classroom instruction on its use and were then given the same three Java programs, requirements, and questions. The experimental group ran the programs through the FX prototype to derive and display their functional behavior, and then answered the questions. Again, all activities were self timed by the subjects and a post-hoc questionnaire was completed.

® CERT is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Both the time required to perform the experimental tasks and the correctness of the answers were measured. The correctness of the answers provided a measure of the ability of both groups to understand the program behaviors and to verify the behaviors against requirements.

The validity of the experimental design and procedures was assessed in four ways. Participants

1. answered the manipulation check question appropriately, which indicated that they understood their task setting (control or experimental group) accurately
2. ranked and rated the difficulty of each of the three programs, named *Algorithm*, *Ordering,* and *Bonus Points*. The participants' assessments were consistent with the experimental design. The shortest, the *Algorithm* program, was ranked as the least difficult of the three, and rated as less difficult than the participants' usual program comprehension tasks. The *Ordering* program is longer than the *Algorithm* program. It was ranked as more difficult to comprehend and rated at about the same level of difficulty as the participants' usual program comprehension tasks. The longest *Bonus Points* program was ranked as the most difficult and rated more difficult than usual comprehension tasks.
3. agreed that the training they received on program comprehension was sufficient to complete the study tasks, and the participants who used the FX prototype strongly agreed that the training they received on its use was sufficient
4. were experienced with program comprehension tasks and were randomly assigned. They had experience in reading and verifying computer code (mean = 5.81 years, range = 2-15 years) and had taken multiple programming classes (mean = 8.5 classes, range = 3-20 classes). In addition, most participants had paid, non-classroom experience in reading and verifying code (mean = 1.34 years, range = 0-5). Tests for equality of variance and means reveals that the control and experimental groups did not differ significantly on any of the three experience variables (see Table 1). These tests are conducted to rule out the possibility that, by chance, more experienced individuals were assigned to either the control or experimental group. (If significantly more experienced individuals had been put into the experimental group, then that higher level of experience would provide a rival explanation for the results.) Because none of these tests have significant results (*p* values are all greater than .10), there is no significant difference between the control and experimental groups in their experience in program comprehension; paid, non-classroom experience; or number of programming classes taken.

Table 1: Comparison of Control and Experimental Groups on Experience

Statistical Test	**Years of Experience in Reading and Verifying Code**	**Number of Programming Classes**	**Years of Paid, Non-Classroom, IT Experience**
Levene's Test for Equality of Variance	F = 2.716 *p* = .112	F = .207 *p* = .654	F = .268 *p* = .610
Test for Independent Samples (t-test for equality of means)	*t* = .937 *p* = .358	*t* = .101 *p* = .921	*t* = -0.333 *p* = .742

4 FX Experimental Results

The results of the experiment strongly demonstrate the significant, positive impact of use of the FX prototype on subjects' performance, measured by time on task and accuracy of program comprehension, as well as on subjects' satisfaction with using the prototype. Subject performance on the experimental tasks (descriptive data are summarized in Table 2) was measured by the amount of time required to complete each task and the accuracy of answers to the questions that tested program comprehension. Subjects reported the time they started and stopped each task, as well as their estimates of how that time was allocated among the following three components of each task:

1. understanding the program requirements and program comprehension questions
2. determining the functionality of the programs
3. recording answers to the program comprehension questions on the form

Table 2: Descriptive Data on Program Comprehension Performance

	Algorithm **Program** (least difficult)		***Ordering*** **Program** (average difficulty)		***Bonus Points*** **Program** (most difficult)	
	Time on Task (minutes)	Percentage correct out of 5 questions	Time on Task (minutes)	Percentage correct out of 10 questions	Time on Task (minutes)	Percentage correct out of 10 questions
Control Group: Traditional Method	Mean = 9.15 Range = 5-12	Mean = 82% Range: 60-100%	Mean = 24.4 Range = 15-35 (3 did not complete)	Mean = 73% Range = 50-90%	Mean = 57 Range = 29-89	Mean = 23% Range = 0-80%
Experimental Group: FX Support	Mean = 5.62 Range = 3-10	Mean = 95% Range = 60-100%	Mean = 14.2 Range = 7-20	Mean = 89% Range = 40-100%	Mean = 13.5 Range = 5-18	Mean = 83% Range = 65-95%

The descriptive data on how subjects divided their time are reported in Table 3. Two rows in this table are highlighted to show the contrast between the two methods of program comprehension: with the FX prototype little or no time was spent determining program functionality, while with the traditional method the majority of time was spent reading and interpreting code to determine its functionality.

The subjects in the experimental group also evaluated the FX prototype on several standard criteria and their measures [Venkatesh 00, Wang 05]. The evaluation criteria, the scale reliability (which identified two items for removal in order to achieve adequate reliability of Cronbach's alpha >.70), and results are shown in Table 4.

Table 3: Mean Percentages of Estimates of How Time Was Spent

	***Algorithm* Program**	***Ordering* Program**	***Bonus Points* Program**
Control Group: Traditional Method			
1. Understanding program requirements and questions	12.4% = 1.1 min	17.8% = 4.3 min	5.0% = 2.9 min
2. Determining program functionality	62.3% = 5.7 min	61.6% = 15.0 min	81.1% = 46 min
3. Recording answers on form	25.3% = 2.3 min	20.6% = 5.0 min	13.9% = 7.9 min
Experimental Group: FX Automation			
1. Understanding program requirements and questions	28.5% = 1.6 min	46.5% = 6.6 min	35.2% = 4.7 min
2. Determining program functionality	0.3% = 0.02 min	0.2% = 0.03 min	0.3% = 0.04 min
3. Recording answers on form	71.2% = 4.0 min	53.3% = 7.7 min	65.3% = 8.8 min

Table 4: Participant Evaluation of the FX Prototype

Evaluation Criterion	**Definition**	**Reliability** (Cronbach's alpha)	**Ratings (n=13)** Rating of 1 = strongly disagree Rating of 5 = strongly agree
Perceived Usefulness	extent to which participants believe that using the FX prototype will enhance job performance	.755	Mean = 4.1 Range = 3-5
Output Quality	participants' assessment of how well the FX prototype performs tasks relevant to the participants' job	.716	Mean = 3.7 Range = 2-5
Technical Utility	participants' assessment of the value, innovativeness, and usefulness of FX prototype	.780	Mean = 4.25 Range = 3-5
Perceived Ease of Use	extent to which participants believe that using the FX prototype is free of effort	.824 (2 items removed)	Mean = 4.5 Range = 3-5
Intention to Use	participants' intention to use the FX prototype in the future	.941	Mean = 4.2 Range = 3-5
Task Support	participants' assessment of the amount of work on task that was not supported by the FX prototype	N/A (1 item, no reliability calculated)	Mean = 2.5 Range = 1-5

Statistical analysis of the study data enables tests of significance of the program comprehension findings. Performance on the three experimental tasks was measured by three dependent variables: accuracy of program comprehension, total time on task, and time required to derive program behavior. Study data were analyzed using analysis of variance (ANOVA), at an alpha level of .05. (An alpha level for a statistical test represents the probability that a significant result from that test is found when in fact there is no significant difference. In experimental research an alpha level of .05 is typically used.) ANOVA is commonly used to test for significant differences between the control and experimental groups. In this study nine tests were run to assess whether or not there were significant differences between groups in the three types of program performance for each of the three study tasks. The results of these tests are shown in Table 5.

Table 5: Results of the ANOVA for Program Comprehension Performance

Performance Measure	***Algorithm* Program** (least difficulty)	***Ordering* Program** (average difficulty)	***Bonus Points* Program** (most difficulty)
Accuracy of Program Comprehension	F = 6.854 *p* = .015	F = 6.251 *p* = .021	F* = 75.489 *p* = .000
Total Time on Task	F = 15.910 *p* = .001	F* = 14.988 *p* = .002	F* = 69.527 *p* = .000
Time Required to Derive Program Behavior	F* = 97.768 *p* = .000	F* = 62.446 *p* = .000	F* = 174.719 *p* = .000

* = asymptotically distributed F statistic from Welch/Brown-Forsythe Robust Test of Equality of Means

A critical assumption of ANOVA is that the control and experimental groups have equal variances. For those tests where the assumption of equal variances based on the Levene statistic is not met (indicated with an asterisk), the Welch/Brown-Forsythe Robust Test of Equality of Means was used, and the asymptotically distributed F statistic reported. In all cases the F statistic represents a ratio of how much the observations vary within each of the groups to how much the observations vary between groups. When the F statistic is near 1 it indicates that there is no statistically significant difference between observations of the control and experimental groups. Larger values of F indicate that the groups' means differ. The *p*-value is a measure of the statistical strength of the differences that are observed. The smaller the *p*-value, the stronger the evidence is of statistical significance. It represents the probability that if we repeated the same experiment we would get results indicating there was no difference between the groups. (In practical terms, very low *p*-values indicate a very low probability that the results were achieved by chance.)

The results reported in Table 5 provide very strong evidence that use of the FX prototype has a positive impact on program comprehension performance: significantly better accuracy of comprehension in significantly less time. The results also show that the FX prototype does in fact automate the derivation of program behavior, since the group using the FX prototype required significantly less time for this part of the task, and, as reported in Table 3, this time averaged less than 1% of experimental subjects' total time on task.

5 Analysis of the Experimental Results

The following observations are based on analysis of the experimental data:

1. **Use of the FX prototype significantly reduces the time required to derive program behavior by truly automating this crucial and time-consuming part of program comprehension**. Subjects who used the FX prototype experienced a significant reduction in the time required for the task of deriving an understanding of program behavior compared with the control group. Whether in the control or experimental groups, subjects had to spend at least some time on understanding the program requirements and the program comprehension questions, and on recording their answers. But the control group subjects using traditional manual methods spent most of their time (between 62% and 81% of total task time) reading and analyzing code to determine program behavior, and this percentage increased as the programs became longer and more difficult. In contrast, the subjects using the FX prototype were able to determine program behavior directly from the prototype output, and thus spent very little time (between .2% and .3% of total task time) on this part of the program comprehension task. Moreover, *the percentage of time subjects took to determine program functionality using the FX prototype did not increase with program length and difficulty, suggesting that improvements will be even more dramatic with further increases in length and difficulty of the programs analyzed*.
2. **Use of the FX prototype significantly improves human performance in program comprehension and verification.** Subjects who used the FX prototype produced far more correct answers to the comprehension questions than the control group; in the case of the longest, most difficult program by a factor of 3.6. They also required far less time to achieve this improved comprehension than the subjects who used traditional methods of reading and inspecting code. While the differences in performance were significant for all programs, the difference in time performance was relatively small for the initial, warm-up task (the *Algorithm* program) but increased as the programs increased in length and difficulty, ultimately resulting in an improvement factor of 4.2 for the *Bonus Points* program. That is, subjects in the experimental group completed this most difficult task in about a fourth of the time required by the control group. Treating productivity as a ratio between task output and input (in this case, output = accurate program comprehension and input = total time on task), *the FX group achieved an improvement in productivity on the order of a factor of 15 for the longest and most difficult program.*
3. **Developers who were trained on and used the FX prototype agree that it is useful, supports the comprehension task, and is easy to use**. *The large improvement in program comprehension for subjects using the FX prototype was achieved with just 45 min-*

utes of instruction, contrasted with years of training and experience in manual program reading and inspection in the control group. As stated by one subject in response to an open-ended question about how well the FX prototype supports program comprehension: “The FX tool was a good tool for verifying computer code. It allows you to get an understanding of what the function actually does without being forced to go through the code line by line. I didn't really read the code because we used the tool.”

In summary, this experiment demonstrates significant advantages for FX technology over traditional methods of code reading and inspection. This objective quantification of the business case for FX will guide future development of the technology. Additional experimental studies are planned to further evaluate human performance in program comprehension with and without FX support. These studies will provide additional feedback for development of FX technology, human interfaces, and operational processes.

References

URLs are valid as of the publication date of this document.

[Hevner 02]	Hevner, A.; Linger, R.; Sobel, A.; & Walton, G. "The Flow-Service-Quality Framework: Unified Engineering for Large-Scale, Adaptive Systems," *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35).* Big Island, Hawaii, Jan. 3-6, 2002. Los Alamitos, CA: IEEE Computer Society Press, 2002.
[Hevner 05]	Hevner, A.; Linger, R.; Collins, R.; Pleszkoch, M.; Prowell, S.; & Walton, G. *The Impact of Function Extraction Technology on Next-Generation Software Engineering* (CMU/SEI-2005-TR-015) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents /05.reports/05tr015.html.
[Hoffman 01]	Hoffman D. & Weiss, D. (eds.). *Software Fundamentals: Collected Papers by David L. Parnas.* Upper Saddle River, NJ: Addison Wesley, 2001.
[Linger 79]	Linger, R.; Mills, H.; & Witt, B. *Structured Programming: Theory and Practice.* Reading, MA: Addison Wesley, 1979.
[McCarthy 63]	McCarthy, J. "A Basis for a Mathematical Theory of Computation," *Computer Programming and Formal Systems.* Edited by P. Braffort & D. Hirschberg. Amsterdam, The Netherlands: North-Holland, 1963.
[Mills 86]	Mills, H.; Linger, R.; & Hevner, A. *Principles of Information System Analysis and Design.* San Diego, CA: Academic Press, 1986.
[Mills 02]	Mills, H. & Linger, R. "Cleanroom Software Engineering," *Encyclopedia of Software Engineering, 2nd ed.* Edited by J. Marciniak. New York, NY: John Wiley & Sons, 2002.
[Pleszkoch 90]	Pleszkoch, M.; Hausler, P.; Hevner, A.; & Linger, R. "Function-Theoretic Principles of Program Understanding," *Proceedings of the 23rd Annual Hawaii International Conference on System Science (HCSS-23),* Kailua-Kona: Hawaii, Jan. 2-5, 1990. Los Alamitos, CA: IEEE Computer Society Press, 1990.

[Pleszkoch 04] Pleszkoch M. & Linger, R. "Improving Network System Security with Function Extraction Technology for Automated Calculation of Program Behavior," *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS-37).* Big Island, Hawaii, Jan. 5-8, 2004. Los Alamitos, CA: IEEE Computer Society Press, 2004.

[Prowell 99] Prowell, S.; Trammell, C.; Linger, R.; & Poore, J. *Cleanroom Software Engineering: Technology and Practice,* Reading, MA: Addison Wesley, 1999.

[Venkatesh 00] Venkatesh, V. & Davis, F. D. "A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies," *Management Science 46,* 2 (Feb. 2000): 186-204.

[Wang 05] Wang, C.; Hsu, Y.; & Fang, W. "Acceptance of Technology with Network Externalities: An Empirical Study," *Journal of Information Technology Theory and Application 6*,4, (2005): 15-28.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 2005	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE The CERT Function Extraction Experiment: Quantifying FX Impact on Software Comprehension and Verification		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Rosann W. Collins; Alan R. Hevner; Gwendolyn H. Walton; & Richard C. Linger		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2005-TN-047
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Function Extraction (FX) is a new, theory-based technology for automated calculation of the functional behavior of software. The CERT Function Extraction experiment was conducted so as to better understand the impact of FX on human comprehension and verification of software and to rigorously quantify the business case for FX technology. This report describes the results of the controlled experiment that was performed to compare traditional manual methods of comprehension with automated behavior computation using an FX prototype. The results of the experiment show a substantial increase in human capabilities for software comprehension and verification using FX technology.

14. SUBJECT TERMS Function Extraction, FX, software program comprehension, software program behavior reading, software program behavior analysis			15. NUMBER OF PAGES 25
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL