MASTER COPY   KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final Report

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Automatically Combining Changes to Software Systems	ARO 117-93
6. AUTHOR(S) Valdis Berzins	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Science Department U.S. Naval Postgradute School Monterey, CA 93943	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution unlimited.	

13. ABSTRACT (Maximum 200 words)

This project has been working to establish a theoretically sound approach to managing changes to software systems via automated methods for combining changes with provable guarantees of correctness. Given a base version of a software system and two different enhanced versions we are seeking to automatically construct a combined version that incorporates both of the enhancements to the base version. Combining changes to a system is a central problem in many software development and maintenance activities, particularly in contexts where several enhancements are developed concurrently.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software change merging, computer aided design, software maintenance, software evolution, concurrent engineering	
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

AUTOMATICALLY COMBINING CHANGES TO SOFTWARE SYSTEMS

FINAL REPORT

Valdis Berzins

JANUARY 1995

U. S. ARMY RESEARCH OFFICE

CONTRACT / GRANT NUMBER ARO 117-93

NAVAL POSTGRADUATE SCHOOL

1. Statement of the Problem Studied

This project has been working to establish a theoretically sound approach to managing changes to software systems via automated methods for combining changes with provable guarantees of correctness. The objectives of the research were

(1) to develop the general mathematics formalizing the semantics of changes to software systems,

(2) to develop algorithms for automatically combining such changes that provide guarantees of correctness when the changes are compatible with each other, and

(3) to detect and locate inconsistencies for correction of requirements when the changes are not compatible.

Given a base version of a software system and two different enhanced versions, we are seeking to automatically construct a combined version that simultaneously incorporates both of the enhancements to the base version. Semantically based methods for combining changes are needed because (1) manual methods are labor intensive and error prone, and (2) conventional tools for combining changes treat software objects as uninterpreted text strings and do not guarantee the integrity of the results.

Combining changes to a system is a central problem in many software development and maintenance activities, particularly in contexts where several enhancements are developed concurrently. Experimental work has established that many software errors can be attributed to the difficulty of understanding interactions between scattered pieces of code [11]. Combining changes to a system is a central problem in many software development and maintenance activities. Software systems are created and evolve in a series of extensions, enhancements, and changes as new requirements are discovered; as existing requirements are extended, reformulated, or dropped; and as system faults are discovered and repaired. This process leads to a branching structure of version histories. Operations for combining changes are needed in all of the following contexts.

(1) Different branches can represent alternative designs for the same enhancement. Automated tools for combining changes can be used to explore alternative choices for decisions in the context of software prototyping and exploratory design. Speed and accuracy provided by tool support can enable exploratory evaluations of design alternatives based on experimental measurements, although these processes are often impractically slow and expensive if done manually, especially when exploring combinations of several interacting design decisions.

(2) Different branches can represent enhancements developed in parallel by different engineers or teams. Semantically based tools for combining changes are useful for combining the results of such parallel efforts. Different people working concurrently on a large software system usually have incomplete knowledge of what the others are doing. Semantically based tools for combining changes are essential for preserving the integrity of such systems, since people can detect inconsistencies only if they have knowledge of a conflicting set of decisions.

(3) Different branches can represent alternative implementations of the system for different operating environments which are derived from a common base version of the system. An enhancement to such a software family can be developed once based on the common root version, and propagated automatically to all of the environment-dependent variations by a tool for combining changes. In the general case, there can be many branches of the development affected by a change, and there can be long chains of indirectly induced modifications, as discussed in [12]. Similar patterns of change propagation occur when a fault in a design decision is discovered only after several subsequent changes have been based on the faulty decision.

2. Summary of the Most Important Results

We have previously investigated the problem of combining programs that compute partial functions [3], which is a simplified version of the problem addressed by this project. We developed the earliest formulation of semantic correctness for merging, in terms of a semantic lattice. This model applies to the special case of compatible extensions to functions, and addresses the problem of merging versions, rather than merging changes to versions. Artificial conflict elements are used to formally locate inconsistencies between versions that conflict. The paper also presents some semantically sound merging methods for functional programs (including recursion but not state changes or loops).

Program modifications and imperative programs were first addressed by [9], using program slicing [13]. This work uses program dependence graphs [8], originally developed for optimizing compilers, to calculate combined changes for flowchart programs with assignments. Investigations of the semantics of slices have shown that the method gives correct results in the cases where it does not report a failure. A weakness of the work is the data-flow approximation used, which does not take into account the semantics of the conditional decisions in the programs. Because of this, the existing program dependence graph algorithms report conflicts between any two changes that potentially affect the same output variables, even if the changes affect disjoint portions of the input space, and therefore cannot interfere with each other. The method also does not have any formal model or representation for inconsistencies, and does not directly provide diagnostic information on failure.

The two approaches outlined above essentially cover the entire state of the art of software merging prior to this project. The main results of the project are models and methods for software merging that combine the complementary strengths of the two approaches described above.

We developed a model of change merging that is a uniform extension of standard denotational semantics [5]. This model handles merging of arbitrary changes to programs, and contains a suitably extended set of conflict elements to support formal location of inconsistencies. Our model is used to determine some general properties of change merging, and in particular to explore the degree to which changes to the components of a functional composition (modules related by data flow relations) can be merged independently. Examples show that this is not possible in the general case, and some special conditions where it is possible are characterized. Our model covers most of the standard constructions of domain theory, including sums, products, function spaces, and two of the three kinds of power domain, and hence applies to a large class of programming languages. The third kind of power domain (the Egli-Milner construction) is shown to have a property that precludes treatment by any of the known formalisms for modeling change merging (i.e., Boolean and Browerian algebras), which indicates fundamental difficulties associated with the interaction between parallel programs and computations that can fail to terminate. This construction is needed if we want the meaning of a parallel program that sometimes works correctly and sometimes diverges to be different from the meaning of a program that always diverges and also different from the meaning of a program that always works correctly.

We also developed a method for semantic change merging based on program meaning functions [4]. This method improves merging accuracy at the expense of computing time. Accuracy is improved in the sense that the method will produce successful and semantically correct merges in cases where the program slicing method will report failures. These cases correspond to the inability of the program slicing approach to recognize disjoint execution path conditions and behavioral equivalences between different algorithms. The meaning function approach should in principle be capable of deriving any semantically valid merge. However, the method can run forever if it is not restricted. Some heuristics to constrain the search for practical use are suggested in the paper. This approach can also produce results in an extended domain that includes representations for conflict elements in programs. In the cases where such conflict elements are produced the method provides diagnostic information to locate particular inconsistencies between program changes that lead to merging failures.

Our theoretical models have been applied to develop and implement a merging method for the prototyping language PSDL [10]. The novel features of this language are hard real-time constraints, parallel computation, and nondeterminism. The merging method is based on an extension of the slicing idea. Correctness of the method depends on a behavioral invariance theorem for slices that was proved relative to a semantic model that captures the nondeterministic and real-time aspects of PSDL programs. An implementation of this method is described in [6, 7].

We have also developed a model for controlling the evolution of a software system [2]. The model is a refinement of earlier work [12] to support the integration of project coordination and configuration management in the context of evolutionary prototyping. The model has been the basis for the design and implementation of an evolution control system for prototypes developed using PSDL and the computer-aided prototyping system CAPS [1]. The functions provided by the evolution control system include computer-aided planning of software evolution steps, automated project scheduling, automated assignment of tasks to designers based on declared management policies, automated versioning of software objects, automated check in and check out of versions from the design database, automated monitoring of progress with respect to deadlines, and decision support for adjusting deadlines if timely completion becomes infeasible.

3. Publications and Technical Reports

(1) V. Berzins, Luqi, A. Yehudai, "Using Transformations in Specification-Based Prototyping", IEEE Transactions on Software Engineering, May 1993, pp. 436-452.

(2) B. Kraemer, Luqi, V. Berzins, "Compositional Semantics of a Real-Time Prototyping Language", IEEE Transactions on Software Engineering, May 1993, pp. 453-477.

(3) D. Dampier, Luqi, V. Berzins, Automated Merging of Software Prototypes, Journal of Systems Integration 4, pp. 33-49, 1994.

(4) V. Berzins, "Software Merge: Semantics of Combining Changes to Programs", to appear in ACM TOPLAS, 1995.

(5) V. Berzins, "Software Merge: Models and Methods for Combining Changes to Programs", Proceedings of the European Conference on Software Engineering, Oct. 1991, p. 221-250, *Lecture Notes in Computer Science*, Vol. 550, Springer-Verlag.

(6) V. Berzins, Luqi, Y. Lee, "Applications and Meaning of Inheritance in Software Specifications", Proceedings of the Hawaii Conference on System Sciences, Koloa, Hawaii, Jan. 7-10, 1992, p. 64-73.

(7) V. Berzins, D. Cooke, Luqi, & M. Tanik, "Workshop on Software Automation", in Proceedings of the IEEE/ACM Second International Conference on Systems Integration, Morristown, NJ, June 15-18, 1992, pp. 720-722.

(8) Dampier, D. and Luqi, "Automated Software Maintenance Using Comprehension and Specification", Proceedings of the 1st Workshop on Program Comprehension, Orlando, Florida, November 9, 1992.

(9) D. Dampier, Luqi, V. Berzins, "Automated Merging of Software Prototypes", Proc. of the Fifth International Conference on Software Engineering and Knowledge Engineering, June 16-18, 1993, San Francisco, pp. 604-611.

(10) S. Badr, Luqi, "A Version and Configuration Model for Software Evolution", Proc. of the Fifth International Conference on Software Engineering and Knowledge Engineering, June 16-18, 1993, San Francisco, pp. 225-227.

(11) Luqi, J. Goguen, "Some Suggestions for Using Formal Methods in Software Development", Proc. AFOSR/ARO/ONR Workshop on Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development, Monterey, CA, Oct. 1993, pp. 7-11.

(12) D. Dampier, V. Berzins, "A Slicing Method for Semantic Based Merging of Software Prototypes", Proc. of the AFOSR/ARO/ONR Workshop on Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development, Oct. 13-15, 1993, Monterey, pp. 22-24.

(13) V. Berzins, "Software Merge: Models and Properties", Proc. of the 6th International Conference on Software Engineering and Knowledge Engineering, Jurmala, Latvia, June 20-23, 1994, pp. 225-232.

(14) Luqi, J. Goguen, V. Berzins, "Formal Support for Software Evolution", Proc. of Monterey Workshop 94, Monterey, CA, Sept. 7-9, 1994, pp. 10-21.

(15) D. Dampier, V. Berzins, "Software Change-Merging in Dynamic Evolution", Proc. of Monterey Workshop 94, Monterey, CA, Sept. 7-9, 1994, pp. 38-41.

(16) S. Badr, V. Berzins, "A Software Evolution Control Model", Proc. of Monterey Workshop 94, Monterey, CA, Sept. 7-9, 1994, pp. 160-171.

(17) V. Berzins, "Software Merge: Semantics of Combining Changes to Programs", Technical Report NPS CS-93-011 Computer Science Department, Naval Postgraduate School, Dec. 1993.

(18) V. Berzins, S. Badr, "Robust Scheduling for Large Projects" Technical Report NPS CS-93-012 Computer Science Department, Naval Postgraduate School, Dec. 1993.

(19) S. Badr and Luqi, "Automation Support for Concurrent Software Engineering" Technical Report NPS-CS-93-013, Computer Science Department, Naval Postgraduate School, December 1993.

4. Participating Scientific Personnel

(1) Valdis Berzins, Computer Science Department, Naval Postgraduate School.

(2) Luqi, Computer Science Department, Naval Postgraduate School.

(3) Salah Badr, Computer Science Department, Naval Postgraduate School, Ph.D., December 1993.

(4) David Dampier, Computer Science Department, Naval Postgraduate School, Ph.D., June 1994.

5. Bibliography

1. S. Badr, ''A Model and Algorithms for a Software Evolution Control System'', Ph.D. Thesis, Computer Science Department, Naval Postgraduate School, Monterey, CA, Dec. 1993.
2. S. Badr and V. Berzins, ''A Software Evolution Control Model'', in *Proc. of Monterey Workshop 94*, Monterey, CA, Sep. 7-9, 1994, 160-171.
3. V. Berzins, ''On Merging Software Extensions'', *Acta Informatica 23*, Fasc. 6 (Nov. 1986), 607-619.
4. V. Berzins, ''Software Merge: Models and Methods'', in *Proceedings of the European Conference on Software Engineering*, Springer-Verlag, Milan, Italy, Oct. 1991, 221-250.
5. V. Berzins, ''Software Merge: Semantics of Combining Changes to Programs'', *to appear in ACM Trans. Prog. Lang and Systems 17* (1995).
6. D. Dampier, Luqi and V. Berzins, ''Automated Merging of Software Prototypes'', *Journal of Systems Integration 4*, 1 (February, 1994), 33-49.

7. D. Dampier, ''A Model for Merging Software Prototypes'', Ph.D. Thesis, Computer Science Department, Naval Postgraduate School, Monterey, CA, June, 1994.

8. J. Ferrante, K. Ottenstein and J. Warren, ''The Program Dependence Graph and its Use in Optimization'', *Trans. Prog. Lang and Systems 9*, 3 (July 1987), 319-349.

9. S. Horwitz, J. Prins and T. Reps, ''Integrating Non-Interfering Versions of Programs'', *Trans. Prog. Lang and Systems 11*, 3 (July 1989), 345-387.

10. B. Kraemer, Luqi and V. Berzins, ''Compositional Semantics of a Real-Time Prototyping Language'', *IEEE Trans. on Software Eng. 19*, 5 (May 1993), 453-477.

11. S. Letovsky and E. Soloway, ''Delocalized Plans and Program Comprehension'', *IEEE Software 3*, 3 (May 1986), 41-49.

12. Luqi, ''A Graph Model for Software Evolution'', *IEEE Trans. on Software Eng. 16*, 8 (Aug. 1990), 917-927.

13. M. Weiser, ''Program Slicing'', *IEEE Trans. on Software Eng. SE-10*, 4 (July 1984), 352-357.