FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Titel der Arbeit

Optionaler Untertitel der Arbeit

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Pretitle Forename Surname Posttitle
Matrikelnummer 0123456

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Pretitle Forename Surname Posttitle
Mitwirkung: Pretitle Forename Surname Posttitle
Pretitle Forename Surname Posttitle
Pretitle Forename Surname Posttitle

Wien, 1. Jänner 2001

Forename Surname	Forename Surname

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Title of the Thesis

Optional Subtitle of the Thesis

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Pretitle Forename Surname Posttitle
Registration Number 0123456

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Pretitle Forename Surname Posttitle
Assistance: Pretitle Forename Surname Posttitle
Pretitle Forename Surname Posttitle
Pretitle Forename Surname Posttitle

Vienna, 1st January, 2001

Forename Surname

Forename Surname

Erklärung zur Verfassung der Arbeit

Pretitle Forename Surname Posttitle
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Jänner 2001

Forename Surname

Danksagung

Ihr Text hier.

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

Enter your text here.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Motivation

- industrial software -> bug -> statistics -> understand(up to 60% [Pig96]) to fix
  - Industrial software, due to its(steady growning) complexity [LB85](need to read)
    structured programming http://dl.acm.org/citation.cfm?id=1243380
  - The mental model of the client -> the specification(formal and informal) -> the mental model of developement team and the programmers -> the actual written behavior.
  - software evolution
    Evelyn Barry , Sandra Slaughter , Chris F. Kemerer, An empirical analysis of software evolution profiles and outcomes, Proceedings of the 20th international conference on Information Systems, p.453-458, December 12-15, 1999, Charlotte, North Carolina, USA
  - maintainance [LS80] [ISO06]
    T. H. Ng , S. C. Cheung , W. K. Chan , Y. T. Yu, Do Maintainers Utilize Deployed Design Patterns Effectively?, Proceedings of the 29th international conference on Software Engineering, p.168-177, May 20-26, 2007
  - In all these transitions information maybe lost and thus some elements of the chain are encouraged to change behavior.
- code gets written one time and read 4 times
- program comprehension
  - strategies [SFM99]
    * top down

    - bottom up
    - knowledgebased
    - systematic and as-needed
    - integrated approaches
  - dynamic analysis as defined by [Bal99] [CZvD+09]
    - actual behavior
    - incomplete view [Bal99]
    - observer effect
      Andrews, J. (1997). Testing using log file analysis: tools, methods, and issues. In Proc. International Conference on Automated Software Engineering (ASE), pages 157âĂŞ 166. IEEE Computer Society Press
    - scalability
      Zaidman, A. (2006). Scalability Solutions for Program Comprehension through Dynamic Analysis. PhD thesis, University of Antwerp
  - static analysis as defined by [Bal99]
    - ...
  - mental model
  - documentation
  - source code level documentation
    Ninus Khamis , Juergen Rilling , RenÃĺ Witte, Assessing the quality factors found in in-line documentation written in natural language: The JavadocMiner, Data & Knowledge Engineering, 87, p.19-40, September, 2013
  - requirements for tools
- debugging -> different kind of paradigms and languages and tools
- concatenative languages -> forth, postscript, factor
- comparisson to oo langs

David Shepherd , Lori Pollock , K. Vijay-Shanker, Case study: supplementing program analysis with natural language analysis to improve a reverse engineering task, Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, p.49-54, June 13-14, 2007, San Diego, California, USA

Martin P. Robillard , Wesley Coelho , Gail C. Murphy, How Effective Developers Investigate Source Code: An Exploratory Study, IEEE Transactions on Software Engineering, v.30 n.12, p.889-903, December 2004

Darren C. Atkinson , William G. Griswold, The design of whole-program analysis tools, Proceedings of the 18th international conference on Software engineering, p.16-27, March 25-29, 1996, Berlin, Germany

1.2 problem statement (which problem should be solved?)

- much work and tools on oo-languages
- not so much on concatenative ~~stack oriented~~ languages
- applicability of oo-methods for concatenative ~~stack oriented~~ languages at the example of forth
- applicability of oo-visualization methods

1.3 aim of the work

- identify important information
- visualization of information
- demo approach

1.4 methodological approach

- qualitative approach(?)
- proposal
- Preliminary evaluations as defined by [CZvD+09]
- outcome is a subjectiv view of the available methods, and proposed enhancements which have been implemented and suggestions of further enhancements

1.5 structure of the work

- summary on the available methods for program comprehension in gforth
- summary and applicability of available methods for other paradigms and languages
- enhancement of existing methods and proposal for further enhancements
- poc

CHAPTER 2

State of the art / analysis of existing approaches

2.1 literature studies

- about program comprehension
  - factoring
  - displayed depth of factoring
  - about debugging
  - dataflow analysis(Backward Analysis)(not sufficient in demo)
    Darren C. Atkinson , William G. Griswold, Implementation Techniques for Efficient Data-Flow Analysis of Large Programs, Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01), p.52, November 07-09, 2001
- about debugging in other paradigms
  - (?)about some tools
- about debugging in stack oriented languages
  - (?)about some tools
- (?)about visualization maybe some examples and tools
  - sequence diagram
  - interaction diagrams (Jacobson, 1992)/ scenario diagrams (Koskimies and MÃűssenbÃűck 1996)

  - information murals (Jerding and Stasko, 1998)
  - polymetric views (Ducasse et al., 2004)
  - hierarchical edge bundling (Holten, 2006)
  - structural and behavioral views of object-oriented program (Kleyn and Gingrich, 1988)
  - matrix visualization and âĂIJexecution patternâĂİ notations [PLVW98] to visualize traces in a scalable manner(De Pauw et al. 1993, 1994, 1998)
  - architecture oriented visualization (Sefika et al. 1996)
  - a continuous sequence diagram, and the âĂIJinformation muralâĂİ (Jerding and Stasko, 1998)
  - architecture with dynamic information (Walker et al. 1998)
  - frequency spectrum analysis (Ball 1999)
- (?)about realtime/interactive vs post mortem

2.2 analysis

- existing methods abstract(abstract like print debugging and stepping and so on)
- applicability for so-languages

2.3 comparison and summary of existing approaches

- existing methods(actual methods)
  - factoring (http://en.wikipedia.org/wiki/Modular_programming https://www.complang.tuwie
    html/Factoring-Tutorial.html http://www.ultratechnology.com/Forth-factors.htm)
  - dump
  - . / type
  - dbg
  - see/ code-see
  - ~~

CHAPTER 3

Methodology

3.1 used concepts

- prototyping
- reading codes
- print-debugging
- step-debugging

3.2 methods and/or models

- prototyping

3.3 languages

- postscript
- forth
- shell script
- c
- m2

3.4 design methods

?

3.5 data models

?

3.6 analysis methods

- reading code
- tail and error

3.7 formalisms

?

CHAPTER 4

Suggested solution/implementation

- kind of an ide
  - interactive program manipulation: state of the system before a word, after a word and by clicking on the word jumping to its definition and there also providing those features
  - stepping debugger mode: simply stepping through the whole code word by word
  - goal-oriented strategy: the definition of an execution scenario such that only the parts of interest of the software system are analyzed (Koenemann and Robertson, 1991; Zaidman, 2006).
  - other data structures and variables should be displayed
  - display of the 'vocabulary'
- proof of concept by enhancement of stepping debugger on forth code level(cause it has turned out to be the fastest and simples approach) by showing additional data: the other stacks

CHAPTER 5

Critical reflection

5.1 comparison with related work

light table ide(js) eclipse ide(java)

5.2 discussion of open issues

- nature of gforth
  - interpretation/compilation mix
  - implementation within the executing system
  - lack of static(and dynamic?) information
- not suitable for performance meassuring
- quantitative data on the effects the enhancement

CHAPTER 6

Summary and future work

summary of what has been done and the subjective conclusion

- see suggested solution
- using a standard data type to store traces
- display of variable content
- display of allocated memory areas
- display of color diff with tooltip of previous values for stacks and memory areas
- better visualization of loops and control structures
- display of the full program as a graph
- customizable inspection depth
- static code analysis
  - stack depth per word
  - type system for forth
  - ...

Bibliography

[Bal99] Thoms Ball. The concept of dynamic analysis. *SIGSOFT Softw. Eng. Notes*, 24(6):216–234, October 1999.

[CZvD+09] Bas Cornelissen, Andy Zaidman, Arie van Deursen, Leon Moonen, and Rainer Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Trans. Softw. Eng.*, 35(5):684–702, September 2009.

[ISO06] ISO. Software engineering – software life cycle processes – maintenance. ISO 14764:2006, International Organization for Standardization, Geneva, Switzerland, 2006.

[LB85] M. M. Lehman and L. A. Belady, editors. *Program Evolution: Processes of Software Change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[LS80] Bennett P. Lientz and E. Burton Swanson. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.

[Pig96] Thomas M. Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[PLVW98] Wim De Pauw, David Lorenz, John Vlissides, and Mark Wegman. Execution patterns in object-oriented visualization. In *In Proceedings Conference on Object-Oriented Technologies and Systems (COOTS âĂŹ98*, pages 219–234, 1998.

[SFM99] M.-A. D. Storey, F. D. Fracchia, and H. A. Müller. Cognitive design elements to support the construction of a mental model during software exploration. *J. Syst. Softw.*, 44(3):171–185, January 1999.