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

200-250 words Enter your text here.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Motivation

- Industrial software, due to its(steady growning) complexity [LB85](need to read) structured programming http://dl.acm.org/citation.cfm?id=1243380
- 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
  code has to be understood [Boe76] in order to make changes or add features [SLea97] integrate somewhere here: software -> bug -> understand(up to 60% [Bas97](is this really related? thorrow reading may be better) [Pig96]) to fix
- program comprehension
  - proper reading as of [Bas97](?)
    systematic approach, strategy may depend on various attributes
  - strategies as stated by [SFM99]
  - dynamic analysis as defined by [Bal99] [CZvD+09]
  - static analysis as defined by [Bal99]
  - mental model(LaToza et al., 2006)
    read: @inproceedingsLieberman:1995:BGC:223904.223969, author = Lieberman, Henry and Fry, Christopher, title = Bridging the Gulf Between Code

and Behavior in Programming, booktitle = Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, series = CHI '95, year = 1995, isbn = 0-201-84705-1, location = Denver, Colorado, USA, pages = 480–486, numpages = 7, url = http://dx.doi.org/10.1145/223904.223969, doi = 10.1145/223904.223969, acmid = 223969, publisher = ACM Press/Addison-Wesley Publishing Co., address = New York, NY, USA,

  - documentation artifacts(requirements to component diagram)
    * source code level documentation
      Ninus Khamis , Juergen Rilling , RenÃľ 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

- comparisson to oo langs
  - paradigm promotes a single shared data structure of high importance and thus may simplify the task of putting all the necesarry runtime information visually together(cite someone who says that its important to have all information visible at every point in time). Although there are several stacks, features like arbitary memory allocation, the focus on stacks is clearly stated.
  - higher abstraction, hard structur boundaries
  - TODO implications from the concatenative nature... ie potential to be more natural to read cause of reverse polish notation
    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

- concatenative languages -> forth, postscript, factor
  - how does software maintenance work in those(evt to future work)
  -

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?)

hypothesis here

- much work and tools on oo- or procedural languages [CZvD+09]
- not so much on concatenative ~~stack oriented~~ languages... nothing in fact, although maybe similarities to procedural
- applicability of oo- and procedural methods for concatenative ~~stack oriented~~ languages at the example of forth
- applicability of oo-visualization methods
- suggestions of (new) methods(lineout style wordlists/words)

1.3 aim of the work

This work aims to better understand how program comprehension is performed in concatenative languages and how it can be made more efficient. The secondary goal is analyse the applicability of existing analysis- and visualization methods and provide modifications to existing visualization methods(and maybe suggestion of new methods). The forth programming language is used as a representative of concatenative languages.

~~demonstration by enhancing the gforth stepping debugger(trace recording, trace visualization, goal-based approach possible)~~

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 implementet
- case study of the implemented enhancement
- suggestions of further enhancements

1.5 structure of the work

At first, the available information of a forth program is identified. The next step is to characterize the information and its necessarity for program comprehension is investigated. The differences of forth and object oriented languages are summarized and then the applicability of existing analysis and visualization methods is presented. The last part of this thesis investigates probable enhancements and modifications to existing methods and proposes new approaches. After the conclusion, the thesis presents further suggestions to support program comprehension and further topics of research in this direction.

CHAPTER 2

State of the art / analysis of existing approaches

This section presents an overview of the work relevant to program comprehension regarding the aim of the work.

2.1 literature studies

software evolution software maintenance program comprehension structured approach and thorrow reading is the most efficient[cite]

2.2 analysis

selected work

2.2.1 program comprehension strategies

about the mental model building

- top down
- bottom up
- knowledgebased
- systematic and as-needed
- integrated approaches

2.2.2 analysis to support program understanding

Several analysis types

dynamic analysis

- about realtime/interactive vs post mortem
- 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
- debugging -> different kind of paradigms and languages and tools
  see @incollectionreiss1993trace, title=Trace-based debugging, author=Reiss, Steven P, booktitle=Automated and Algorithmic Debugging, pages=305–314, year=1993, publisher=Springer
- 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

static analysis

complete view no actual data present

2.2.3 applicability to concatenative languages

existing methods abstract(abstract like print debugging and stepping and so on) furthermore the abstraction of all those methods mentioned above applicability for concatenative languages

2.3 visualization to support program understanding maybe some examples(and tools)

- sequence diagram

- circular diagram and interactive interaction sequance diagram [Cor09]
- interaction diagrams (Jacobson, 1992)/ scenario diagrams (Koskimies and MÃűssenbÃűck 1996)
- information murals (Jerding and Stasko, 1998)
- polymetric views (Ducasse et al., 2004)
- fisheye views (suggested by George W. Furnas, 1986, and formulated by [SM96] and [SB94])
- 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)

2.4 comparison and summary of existing approaches

existing approaches for gforth/forth

- kgforth http://sourceforge.net/projects/kgforth/
- existing methods(actual methods)
  - factoring (http://en.wikipedia.org/wiki/Modular_programming https://www.complang.tuwien.ac.at/html/Factoring-Tutorial.html http://www.ultratechnology.com/Forth-factors.htm) has to be considered during initial development
  - dump
  - . / type
  - dbg
  - see/ code-see
  - ~~

CHAPTER 3

Methodology

- case study(maybe exploratory)
- prototype
- sketches
- trying to understand programs developed withing stackbased languages vl?

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~~ development environment
light table ide(js) continuous reverse engineering idea of [MJS+00] to provide immediate resonse of the systems output... although probably not applicable or very time consuming in setup(or not more than integration testing...) for most industrial scale software
eclipse ide(java)

4.1 suggested solution

- emphasis on on comprehension code while writing. factoring suggestion, documentation, aliases(same code with multiple aliases to read more natural at different points in programs), expressive naming, hard to generalize cause of the flexibility the language provides

- adequate search and corss reference facilities to support systematical investigation to benefit from effective program understanding as stated by [RCM04]

- display of the 'vocabulary' [cite moore: remember all the words]

- other data structures and variables should be displayed

  - memory maybe like [Rei95] or [AKG+10] but since there is no underlying object orientation and no standardized oo system this would be hard do accomplish

  - fisheye or word cloud like display(tree or sugiyama as of [SWFM97])

- interactive program manipulation: state of the system before a word, after a word and by clicking on the word jumping to its definition or inserting it 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).
- code analysis and visualization facilities see chapter 2 TODO

4.2 implementation

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

? is there any? maybe the modifications to oo methods? or listing of the methods which did work and those which did not

kgforth in some way

5.2 discussion of open issues

- not scaling well cause of limited screen real estate and thus the need to scroll
- not scalign well cause of unpredictable stack height(maybe show only depth according to stack effect comment)
- nature of gforth
  - interpretation/compilation mix(how to integrate the adhook changes between modes '[]')
  - implementation within the executing system
  - lack of dynamic information(return stack add -> wordname heuristic)
- not suitable for performance meassuring cause debugger...
- quantitative data on the effects the enhancement needed

CHAPTER 6

Summary and future work

summary of what has been done and the subjective conclusion

- ide
- 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) is this even possible?
- (display of the full program as a graph) is this even possible?
- (customizable inspection depth) ?
- static code analysis
  - stack depth per word
  - type system for forth
  - ...

conclusion like what i contributed to the community!!

work on program comprehension of concatenative languages good overview of the field [CDPC11] and [Cor09]

Bibliography

[AKG+10] Edward E. Aftandilian, Sean Kelley, Connor Gramazio, Nathan Ricci, Sara L. Su, and Samuel Z. Guyer. Heapviz: Interactive heap visualization for program understanding and debugging. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, pages 53–62, New York, NY, USA, 2010. ACM.

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

[Bas97] Victor R. Basili. Evolving and packaging reading technologies. *J. Syst. Softw.*, 38(1):3–12, July 1997.

[Boe76] B. W. Boehm. Software engineering. *IEEE Trans. Comput.*, 25(12):1226–1241, December 1976.

[CDPC11] Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. Achievements and challenges in software reverse engineering. *Commun. ACM*, 54(4):142–151, April 2011.

[Cor09] Bas Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Wohrmann Print Service, 2009.

[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.

[MJS+00] Hausi A. Müller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley, and Kenny Wong. Reverse engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 47–60, New York, NY, USA, 2000. ACM.

[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.

[RCM04] Martin P. Robillard, Wesley Coelho, and Gail C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Trans. Softw. Eng.*, 30(12):889–903, December 2004.

[Rei95] S. P. Reiss. *Visualization for Software Engineering – Programming Environments.* 1995.

[SB94] Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73–83, December 1994.

[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.

[SLea97] Janice Singer, Timothy C. Lethbridge, and et al. An examination of software engineering work practices, 1997.

[SM96] Margaret-Anne D. Storey and Hausi A. Müller. Graph layout adjustment strategies. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 487–499, London, UK, UK, 1996. Springer-Verlag.

[SWFM97] M.-A. D. Storey, K. Wong, F. D. Fracchia, and H. A. Mueller. On integrating visualization techniques for effective software exploration. In *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, INFOVIS '97, pages 38–, Washington, DC, USA, 1997. IEEE Computer Society.