UNIVERSIDADE FEDERAL DA BAHIA
Disciplina: *Metodologia da Pesquisa* Turma: 2014
Cursos: PPGM/PEI/PGCOMP
Profs.: Luciano Oliveira, Karen Pontes, Christina von Flach

Aluno: Joenio Marques da Costa

1º Trabalho – orientações e procedimentos

1) A partir do Portal de Periódicos da CAPES (http://www.periodicos.capes.gov.br) execute as seguintes tarefas:

a) Indique 5 palavras-chave em português e suas correspondentes em inglês relacionadas ao tema da sua tese/dissertação.

(1) static code analysis, (2) software visualization.

(utilizei apenas 2 palavras-chave ao invés de 5 pois o resultado da busca com 5 palavras ficou muito restrito)

b) Mostre o resultado da pesquisa com as palavras-chave escolhidas no *Compendex Engineering Index* (acesse o portal via coleções, aplique os filtros). Organize o resultado por ordem relevância (da maior para a menor, até os 100 primeiros). Inclua os resumos e palavras-chave (na versão digital).

***107** artigos encontrados com a seguinte string de busca:*

(((static code analysis) WN All fields) AND ((software visualization) WN All fields))

*(os **100** artigos ordenados por relevância estão anexados ao final desde documento)*

c) De forma similar, mostre o resultado da pesquisa com as palavras-chave escolhidas via o sistema de busca integrada do portal. Organize o resultado por ordem relevância.

Realizei a mesma pesquisa no portal Periodicos retornou 5393 resultados, não há opção no portal Periodicos de exportar a busca. Em anexo ao final deste documento um printscreen da tela.

d) Liste os 5 periódicos indexados (pelo menos 3 deverão ser internacionais) mais importantes para o seu tema de pesquisa, indicando o fator de impacto, a(s) área(s) e a classificação do mesmo no sistema Qualis da CAPES, Engenharia III ou Ciência da Computação (a depender de seu curso).

01 An effective visual system for static analysis of source code

02 Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey

03 Fast analysis of source code in C and C++

04 Seesoft--A tool for visualizing line oriented software statistics

05 Maintenance tools

Pesquisei na Web Qualis estando na rede UFBA e não encontrei resultado para os artigos acima.

(a lista dos artigos com detalhes estão anexados ao final deste documento)

e) Faça uma pesquisa bibliográfica em no mínimo de 8 artigos (sendo pelo menos 5 internacionais), que deverá ter as seguintes características: 1 artigo clássico de revisão, que obrigatoriamente todos os pesquisadores trabalhando no tema devem conhecer e citar; 2 artigos com forte base teórica no tema, de autores diferentes; 5 artigos bem recentes (publicados há menos de 3 anos) e fortemente relacionados ao seu tema de tese/dissertação.

01) Design Suite: Towards an Open Scientific Investigation Environment for Software Architecture Recovery

02) Evolution of Open Source Software Systems – A Large-Scale Investigation

03) How do committees invent?

04) Mining Version Histories to Guide Software Changes

05) On the Criteria To Be Used in Decomposing Systems into Modules

06) Técnicas de Visualização para Avaliação e Melhoria de Qualidade de Software Livre e Aberto

07) The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations

08) XFlow: An Extensible Tool for Empirical Analysis of Software Systems Evolution

f) Faça um texto seu em português (de 3.000 a 4.000 caracteres contados sem espaço), que resuma e sistematize as idéias principais dos artigos selecionados na pesquisa bibliográfica. Observação: o texto deverá manter uma coerência lógica entre os artigos analisados, evitando-se apenas juntar resumos de cada artigo separadamente.

A análise e estudo da evolução de sistemas comerciais é uma área há muito tempo pesquisada pelos centros de pesquisas em engenharia de software, isto proporcionou a criação e evolução de diversas técnicas e frameworks para este tipo de análise. O mesmo não ocorreu no contexto de sistemas "open source" que carecem de experiências neste sentido, dessa forma um estudo foi feito com o intuito de utilizar os resultados de anos de pesquisas em sistemas comerciais em novas pesquisas focadas em sistemas "open source", com o objetivo de comparar seus resultados e identificar se existem diferenças significativas entre os processos de desenvolvimento de cada um. Isto irá proporcionar um entendimento maior da evolução de sistemas "open source", além de jogar luz sobre como estes sistemas são desenvolvidos. Sabe-se ainda muito pouco sobre alguns aspectos do desenvolvimento de softwares, especialmente sobre como desenvolvedores tomam decisões de design relativas a modularização por exemplo. Durante o design e o desenvolvimento de software comumente nada é dito sobre quais critérios devem ser utilizados para dividir um sistema em módulos, cada desenvolvedor toma decisões inidivuais. Estudos foram feitos com o objetivo de identificar tais critérios e tentou-se mostrar alternativas à simples estratégia de pensar módulos de sistemas como simples fluxogramas de forma que ao invés de começar com uma lista de dificuldades de decisões de design desenha-se cada módulo como um conjunto de subrotinas relacionadas. Neste caminho entre o processo de desenvolvimento e o produto final desenvolvido estudos sobre evolucao de software tem tentado entender a relacao entre software e seus desenvolvedores. Tais estudos usualmente requerem ferramentas de suporte para lidar com grandes volumes de dados, estes dados são complexos e precisam ser coletados, processados e analizados. Inúmeras ferramentas tem sido propostas, mas a maioria delas oferecem suporte limitado ao estudo da evolucao de software que considerem aspectos técnicos e sociais em conjunto. Assim nasce a ferramenta XFlow, uma ferramenta extensivel para analise empirica da evolucao de software considerando aspectos técnicos e sociais, esta ferramenta coleta dados de sistema de gerenciamento de configuracoes, precessa tais informações computando metricas e finalmente apresenta poderosas e interativas visualizacoes. A ferramenta ainda destaca-se por possuir habilidades para prover informações úteis para formular e testar hipóteses.

g) Traduza o resumo para o inglês (de 3.000 a 4.000 caracteres, sem espaço).

The analysis and study of the evolution of commercial systems is an area long time studied by research centers in software engineering, this led to the creation and evolution of various techniques and frameworks for this type of analysis. This doesn't occur in the context of the "open source" systens that lack of experience in this sense, thus a study was done in order to use the results of years of research in commercial systems in a new research focused on "open source" systems, with the goal of compare their results and to identify whether there are significant differences between the processes of development of each. This will provide a greater understanding of the evolution of "open source" systems, and shed light about how these systems are developed. Very little is known about some aspects of software development, especially on how developers make design decisions regarding modularization by instance. During the design and development of software is commonly nothing said about which criteria should be used to divide a system into modules, each developer takes inidivuais decisions. Studies were made to identify such criteria and we tried to show alternatives to simple strategy of thinking systems modules as simple flowcharts so that instead of starting with a list of difficulties decisions design each module is drawn as a set of related subroutines. In this way between the development process and the final product developed studies on evolution of software has trying to understand the relationship between software and its developers. Such studies usually require support tools to handle large volumes of data, these data are complex and need to be collected, processed and analyzed. numerous tools have been proposed, but most of them offer limited support the study of the evolution of software to consider technical and social aspects together. Thus was born the XFlow tool, a tool for extensible empirical analysis of the evolution of software considering technical and social aspects, this tool collects data management system configurations, such information precesses computing metrics and finally presents powerful and interactive visualizations. The tool still stands out to possess skills to provide useful information for formulating and testing hypotheses.

2) Indique pelo menos 1 grupo de referência no Brasil e 4 grupos de referência no exterior que estejam trabalhando no tema de sua tese/dissertação. Identifique o pesquisador líder em cada um desses grupos, bem como teses e dissertações orientadas. Avalie a possibilidade de realizar o doutorado sanduíche no exterior (apenas para os alunos de doutorado), levando em conta o tempo previsto (6 a 12 meses) e a proficiência no idioma, e indique a universidade e grupo de pesquisa de sua preferência, justificando sua escolha.

***(1)** Scientific Visualization and Computer Graphics (SVCG) na Universidade de Groningen (Holanda), **(2)** Grupo de Engenharia de Software da Universidade de Trier (Alemanha), **(3)** REVEAL (Reverse Engineering, Visualization, Evolution Analysis Lab) da Universidade de Lugano (Suíca) e, **(4)** Laboratório de Engenharia de Software da Universidade Federal da Bahia (UFBA) no Brasil*

3) Indique pelo menos 1 congresso nacional, 3 congressos internacionais fortemente relacionados a seu tema de tese/dissertação.

(1) *ACM Symposium on Software Visualization (SOFTVIS);* ***(2)*** *IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT);* ***(3)*** *Program Visualization Workshop (PVW);* ***(4)*** *Simpósio Brasileiro de Engenharia de Software (SBES).*
.

4) Indique patentes ou registro de direitos autorais relacionados ao seu tema de tese/dissertação. Justifique, caso não se aplique ao seu caso.

A minha pesquisa é direcionada para comunidade de softwares e ferramentas livres, portanto patentes não é comum e pouco usual neste meio.

Prazo e instruções para entrega:

22/04/2014 - Entrega do trabalho impresso, frente e verso, (1 cópia) nas secretarias dos respectivos programs: PEI ou PPGM ou PGCOMP. A versão impressa pode prescindir dos resumos dos artigos (são suficientes o título, autores e sua afiliação, publicação e palavras-chave). Uma versão digital completa (com os resumos) deve também ser gerada. Esta deve ser em formato pdf e deve ser postada para os respectivos professores: lrebouca@ufba.br (PPGM), karenpontes@ufba.br (PEI), flach@dcc.ufba.br (PGCOMP) e sob o Assunto: 1º Trabalho Metodologia da Pesquisa.

A correção gramatical, o estilo da escrita, a adequação do vocabulário, a formatação do texto e a correção no processo de envio do documento também serão avaliados.

1. Using static code analysis tools to increase source code maintainability

Novak, J. (1); Hericko, M. (1)
Source: *MIPRO 2009 - 32nd International Convention Proceedings: Telecommunications and Information*, v 2, p 145-148, 2009, *MIPRO 2009 - 32nd International Convention Proceedings: Telecommunications and Information*; **Conference:** 32nd International Convention Proceedings: Microelectronics, Electronics and Electronic Technology, MEET and Grid and Visualizations Systems, GVS, May 25, 2009 - May 29, 2009; **Sponsor:** Croatian Electricity Company (HEP); Ericsson Nikola Tesla, Croatia; et al; Siemens, Croatia; T-Croatian Telecom; VIPnet, Croatia; **Publisher:** Croatian Society for Information and Communication Technology
Author affiliation: (1) University of Maribor, Smetanova ul. 17, 2000 Maribor, Croatia
Abstract: There are a lot of static code analysis tools to automatically find program faults. These tools can analyze software without actually executing the programs. Code analysis tools can play an essential role in creating secure and reliable software. They can help catch common coding mistakes such as buffer overflow, cross-site scripting, Structured Query Language (SQL) injections, and a variety of race conditions. They also can help write code which is easier to maintain. The purpose of this paper is to present tools for static code analysis on C# language and how can be these tools used to help write more maintainable code. In experimental part we evaluate code before and after use of tools with maintainability index defined by Oman and Hagemeister. (9 refs)
Main heading: Tools
Controlled terms: Maintainability - Microelectronics - Query languages
Uncontrolled terms: Buffer overflows - Code analysis - Cross site scripting - Source codes - Static code analysis - Static code analysis tools - Structured query languages
Classification Code: 603 Machine Tools - 605 Small Tools and Hardware - 713 Electronic Circuits - 714 Electronic Components and Tubes - 723.3 Database Systems - 913.5 Maintenance
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

2. An effective visual system for static analysis of source code

Wan, Ying (1); Tan, Chuanqi (2); Wang, Zhigang (1, 2); Wang, Guoqiang (1); Hong, Xiaojin (1)
Source: *Advanced Materials Research*, v 433-440, p 5453-5458, 2012, *Materials Science and Information Technology, MSIT2011*; **ISSN:** 10226680; **ISBN-13:** 9783037853191; **DOI:** 10.4028/www.scientific.net/AMR.433-440.5453; **Conference:** 2011 International Conference on Material Science and Information Technology, MSIT2011, September 16, 2011 - September 18, 2011; **Sponsor:** Singapore Institute of Electronics; **Publisher:** Trans Tech Publications
Author affiliation: (1) Lab of Computer Network Defense Technology, Beijing Institute of Technology, China (2) School of Mechatronical Engineering, Beijing Institute of Technology, China
Abstract: In the software development lifecycle, code static analysis takes an important part in building secure software. To help discover the potential security issues in source code, large numbers of static analysis tools are developed. But the results generated by them display in the form of pure text, so it is time-consuming for developers to analyze these text messages, and it is difficult for developers to concentrate on the most interesting defects in huge data. In this paper, we have developed a visualization system oriented to Java source code, which presents the results in graphics from a developer's point of view, to help developers to analysis code defects. A novel layout is proposed to visualize software source code in a hierarchy way, which shows the physical structure of the software. A visual overview and powerful interaction is provided in this system which allows the developer to focus on the most pressing defects within huge volumes of source code. © (2012) Trans Tech Publications, Switzerland. (9 refs)
Main heading: Static analysis
Controlled terms: Building codes - Codes (symbols) - Computer programming languages - Defects - Information technology - Software design - Telephone systems - Visualization
Uncontrolled terms: In-buildings - Java source codes - Physical structures - Secure software - Security issues - Software development life cycle - Software source codes - Source codes - Text messages - Visual systems - Visualization system
Classification Code: 903 Information Science - 902.1 Engineering Graphics - 723 Computer Software, Data Handling and Applications - 951 Materials Science - 718.1 Telephone Systems and Equipment - 403 Urban and Regional Planning and Development - 402 Buildings and Towers - 423 Non Mechanical Properties and Tests of Building Materials
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

3. Software metrics in static program analysis

Vogelsang, Andreas (1); Fehnker, Ansgar (2); Huuck, Ralf (2); Reif, Wolfgang (3)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 6447 LNCS, p 485-500, 2010, *Formal Methods and Software Engineering - 12th International Conference on Formal Engineering Methods, ICFEM 2010, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3642169007, **ISBN-13:** 9783642169007; **DOI:** 10.1007/978-3-642-16901-4_32; **Conference:** 12th International Conference on Formal Engineering Methods, ICFEM 2010, November 17, 2010 - November 19, 2010; **Publisher:** Springer Verlag
Author affiliation: (1) Fakultät für Informatik, Technische Universität München, Boltzmannstr. 3, Garching b., München 85748, Germany (2) National ICT Australia Ltd. (NICTA), University of New South Wales, Locked Bag 6016, Sydney, NSW 1466, Australia (3) Lehrstuhl für Softwaretechnik und Programmiersprachen, Universität Augsburg, Universtitätsstrasse 14, Augsburg 86135, Germany
Abstract: Software metrics play an important role in the management of professional software projects. Metrics are used, e.g., to track development progress, to measure restructuring impact and to estimate code quality. They are most beneficial if they can be computed continuously at development time. This work presents a framework and an implementation for integrating metric computations into static program analysis. The contributions are a language and formal semantics for user-definable metrics, an implementation and integration in the existing static analysis tool , and a user-definable visualization approach to display metrics results. Moreover, we report our experiences on a case study of a popular open source code base. © 2010 Springer-Verlag Berlin Heidelberg. (16 refs)
Main heading: Formal methods
Controlled terms: Computer software maintenance - Computer software selection and evaluation - Integration - Quality control - Semantics - Static analysis - Visualization
Uncontrolled terms: Code quality - Development time - Formal Semantics - Open-source code - Professional software - Software maintenance - software metrics - Software Quality - Static program analysis - Track development
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics - 903.2 Information Dissemination - 913.3 Quality Assurance and Control - 921.2 Calculus
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

4. ClonEvol: Visualizing software evolution with code clones

Hanjalic, Avdo (1)
Source: *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013*, 2013, *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013*; **ISBN-13:** 9781479914579; **DOI:** 10.1109/VISSOFT.2013.6650525; **Article number:** 6650525; **Conference:** 2013 1st IEEE Working Conference on Software Visualization, VISSOFT 2013, September 27, 2013 - September 28, 2013; **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computing Science, University of Groningen, Netherlands
Abstract: We present ClonEvol, a visual analysis tool that assists in obtaining insight into the state and the evolution of a C/C++/Java source code base on project, file and scope level. ClonEvol combines information obtained from the software versioning system and contents of files that change between versions; The tool operates as tool-chain of Subversion (SVN), Doxygen (applied as static analyzer) and Simian as code duplication detector. The consolidated information is presented to the user in an interactive visual manner. The focus of the presented tool lies on scalability (in time and space) concerning data acquisition, data processing and visualization, and ease of use. The visualization is approached by using a (mirrored) radial tree to show the file and scope structures, complemented with hierarchically bundled edges that show clone relations. We demonstrate the use of ClonEvol on a real world code base. © 2013 IEEE. (11 refs)
Main heading: Cloning
Controlled terms: Data acquisition - Tools - Visualization
Uncontrolled terms: Code clone - Code duplication - Software Evolution - Software evolution analysis - Software visualization - Static analyzers - Versioning systems - Visual analysis
Classification Code: 461.8.1 Genetic Engineering - 603 Machine Tools - 605 Small Tools and Hardware - 723.2 Data Processing and Image Processing - 902.1 Engineering Graphics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

5. Combining static and dynamic data in code visualization

Eng, David (1)

Source: *ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, p 43-50, 2002; **DOI:** 10.1145/586094.586105; **Conference:** Proceedings of the 4th 2002 ACM SIGPLAN SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, November 18, 2002 - November 19, 2002; **Sponsor:** Association for Computing and Machinery; Special Interest Group om Programming Languages; Special Interest Group on Software Engineering; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Sable Research Group, McGill University, Montreal, Que., H3A 2A7, Canada
Abstract: The task of developing, tuning, and debugging compiler optimizations is a difficult one which can be facilitated by software visualization. There are many characteristics of the code which must be considered when studying the kinds of optimizations which can be performed. Both static data collected at compile-time and dynamic runtime data can reveal opportunities for optimization and affect code transformations. In order to expose the behavior of such complex systems, visualizations should include as much information as possible and accommodate the different sources from which this information is acquired. This paper presents a visualization framework designed to address these issues. The framework is based on a new, extensible language called JIL which provides a common format for encapsulating intermediate representations and associating them with compile-time and runtime data.
We present new contributions which extend existing compiler and profiling frameworks, allowing them to export the intermediate languages, analysis results, and code metadata they collect as JIL documents. Visualization interfaces can then combine the JIL data from separate tools, exposing both static and dynamic characteristics of the underlying code. We present such an interface in the form of a new web-based visualizer, allowing JIL documents to be visualized online in a portable, customizable interface. (18 refs)
Main heading: Program compilers
Controlled terms: Computer software - Data acquisition - Interfaces (computer) - Java programming language - Performance - Program debugging - Visualization
Uncontrolled terms: Dynamic data
Classification Code: 722.2 Computer Peripheral Equipment - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.2 Data Processing and Image Processing
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

6. Visualization tools for understanding a complex code from a real application

Campos, Fernanda (1); Cortazar, Esteban (1); Eterovic, Yadran (1); Ramirez, Leonardo (2); Tejos, Cristian (2); Irarrazaval, Pablo (2)
Source: *22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE 2009*, p 284-291, 2009, *22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE 2009*; **ISBN-13:** 9781615676668; **Conference:** 22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE 2009, November 4, 2009 - November 6, 2009; **Publisher:** International Society for Computers and Their Applications
Author affiliation: (1) Department of Computer Science, Pontificia Universidad Católica de Chile, Chile (2) Department of Electrical Engineering, Biomedical Imaging Center, Pontificia Universidad Católica de Chile, Chile
Abstract: Research in Magnetic Resonance Imaging (MRI) requires researchers to make changes to the software that controls the scanner. Thus, the first challenge faced by MRI researchers is to understand the scanner's code to find the places where the changes must be made, and to make sure that the changes will not produce undesired effects in other parts of the code. At the Biomedical Imaging Center at Pontificia Universidad Cato´lica de Chile, we have been developing visualization tools for analysing the Philips scanner's code, and for storing and sharing the know-how thus acquired. We devised the tools to help MRI researchers identify the code's main concerns and understand how these concerns are represented and handled within the code. In this paper we report on the tools' functionalities; e.g., finding the places where each object or function is used, analyzing relationships among objects, storing comments to objects and functions, identifying which functions are executed and in which order, representing this information as a tree, searching this tree for a given function, and highlighting the differences between the trees representing two different executions. Initial tests show that these tools effectively support MRI researchers in their task of understanding the scanner's code. (11 refs)
Main heading: Tools
Controlled terms: Computer applications - Forestry - Magnetic resonance imaging - Medical imaging - Research - Scanning - Technology transfer - Visualization
Uncontrolled terms: Biomedical imaging - Code analysis - Complex codes - Philips - Program comprehension - Real applications - Static code analysis - Visualization tools

Classification Code: 901.4 Impact of Technology on Society - 901.3 Engineering Research - 821.0 Woodlands and Forestry - 746 Imaging Techniques - 902.1 Engineering Graphics - 741.3 Optical Devices and Systems - 723.5 Computer Applications - 605 Small Tools and Hardware - 603 Machine Tools - 741 Light, Optics and Optical Devices
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

7. Three-dimensional visualization tool for software fault analysis of a distributed system

Amari, Haruo (1); Okada, Mikio (1)
Source: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, v 4, p IV-194 - IV-199, 1999; **ISSN:** 08843627; **DOI:** 10.1109/ICSMC.1999.812399; **Conference:** 1999 IEEE International Conference on Systems, Man, and Cybernetics 'Human Communication and Cybernetics', October 12, 1999 - October 15, 1999; **Sponsor:** IEEE (SMC); SCJ; SICE; RSJ; JSME; **Publisher:** IEEE
Author affiliation: (1) Tokyo Electric Power Co, Yokohama, Japan
Abstract: In this paper we propose a software visualization tool named Software Visualization Supporting Space (SVSS) utilizing three-dimensional (3D) graphics in order to detect and analyze software faults in a large-scale distributed system. When a failure occurs in a software testing process, it is necessary for fault detection to inspect a large number of software processes from various viewpoints. 3D graphical representations allow a greater quantity of data as well as complex structures and relationships between components to be displayed on the screen effectively. SVSS can analyze the source code regularly to generate structural data of the target software. Moreover, it can also analyze the behavior of the processes using trace data obtained automatically by embedding a trace data acquisition function in the target machine. The results of these analysis functions can be visualized collectively on a common display so that the developer can trace execution flows and static connections simultaneously. The work reduction ratio using SVSS was estimated by sampling faults that had occurred in practical development of a distributed power control system. (11 refs)
Main heading: Computer aided software engineering
Controlled terms: Computer software - Data acquisition - Distributed parameter control systems - Electric current control - Electric power systems - Reliability - Three dimensional computer graphics
Uncontrolled terms: Distributed power control system - Software fault analysis - Software visualization supporting space
Classification Code: 706.1 Electric Power Systems - 723.1 Computer Programming - 723.2 Data Processing and Image Processing - 723.5 Computer Applications - 731.1 Control Systems - 731.3 Specific Variables Control
Treatment: Applications (APP) - Theoretical (THR) - Experimental (EXP)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

8. The Solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product

Reniers, Dennie (1); Voinea, Lucian (1); Ersoy, Ozan (2); Telea, Alexandru (2)
Source: *Science of Computer Programming*, v 79, p 224-240, January 1, 2014; **ISSN:** 01676423; **DOI:** 10.1016/j.scico.2012.05.002; **Publisher:** Elsevier
Author affiliation: (1) SolidSource BV, Eindhoven, Netherlands (2) Institute Johann Bernoulli, University of Groningen, Netherlands
Abstract: Software visual analytics (SVA) tools combine static program analysis and fact extraction with information visualization to support program comprehension. However, building efficient and effective SVA tools is highly challenging, as it involves extensive software development in program analysis, graphics, information visualization, and interaction. We present a SVA toolset for software maintenance, and detail two of its components which target software structure, metrics and code duplication. We illustrate the toolset's usage for constructing software visualizations with examples in education, research, and industrial contexts. We discuss the design evolution from research prototypes to integrated, scalable, and easy-to-use products, and present several guidelines for the development of efficient and effective SVA solutions. © 2011 Elsevier B.V. All rights reserved. (93 refs)
Main heading: Static analysis
Controlled terms: Industrial research - Information analysis - Information systems - Software engineering - Tools - Visualization
Uncontrolled terms: Easy-to-use products - Information visualization - Program structures - Research prototype - Software structures - Software visualization - Static program analysis - Visual tools

Classification Code: 603 Machine Tools - 605 Small Tools and Hardware - 723.1 Computer Programming - 901.3 Engineering Research - 902.1 Engineering Graphics - 903 Information Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

9. Magnify - A new tool for software visualization

Bartoszuk, Cezary (1); Timoszuk, Grzegorz (1); Dabrowski, Robert (1); Stencel, Krzysztof (1)
Source: *2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013*, p 1485-1488, 2013, *2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013*; **ISBN-13:** 9781467344715; **Article number:** 6644213; **Conference:** 2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013, September 8, 2013 - September 11, 2013; **Sponsor:** Ministry of Science and Higher Eduction; Intel; **Publisher:** IEEE Computer Society
Author affiliation: (1) Institute of Informatics, University of Warsaw, Banacha 2, Warsaw 02-097, Poland
Abstract: Modern software systems are inherently complex. Their maintenance is hardly possible without precise up-to-date documentation. It is often tricky to document dependencies among software components by only looking at the raw source code. We address these issues by researching new software analysis and visualization tools. In this paper we focus on software visualisation. Magnify is our new tool that performs static analysis and visualization of software. It parses the source code, identifies dependencies between code units and records all the collected information in a repository based on a language-independent graph-based data model. Nodes of the graph correspond to program entities of disparate granularity: methods, classes, packages etc. Edges represent dependencies and hierarchical structure. We use colours to reflect the quality, sizes to display the importance of artefacts, density of connections to portray the coupling. This kind of visualization gives bird's-eye view of the source code. It is always up to date, since the tool generates it automatically from the current revision of software. In this paper we discuss the design of the tool and present visualizations of sample open-source Java projects of various sizes. © 2013 Polish Information Processing Society. (20 refs)
Main heading: Static analysis
Controlled terms: Computer programming languages - Computer science - Information systems - Tools - Visualization
Uncontrolled terms: Bird's eye view - Hierarchical structures - Software analysis - Software component - Software systems - Software visualisation - Software visualization - Visualization tools
Classification Code: 903.2 Information Dissemination - 902.1 Engineering Graphics - 723 Computer Software, Data Handling and Applications - 722 Computer Systems and Equipment - 721 Computer Circuits and Logic Elements - 605 Small Tools and Hardware - 603 Machine Tools
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

10. A combined software reconnaissance & static analysis eclipse visualisation plug-in

Cleary, Brendan (1); Le Gear, Andrew (1); Exton, Chris (1); Buckley, Jim (1)
Source: *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, p 121-122, 2005, *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*; **ISBN-10:** 0780395409, **ISBN-13:** 9780780395404; **DOI:** 10.1109/VISSOF.2005.1684319; **Article number:** 1684319; **Conference:** 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2005, September 25, 2005 - September 25, 2005; **Sponsor:** IEEE Computer Society; **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computer Science and Information Systems, University of Limerick, Ireland
Abstract: Software reconnaissance is a dynamic analysis technique which can aid in the mapping between program features and code that implements those features. In a previous case study we have shown how we were able to combine software reconnaissance with a static data analysis to a derive reuse perspective from an existing system. In this paper we report on the tool support, in the form of an eclipse plug-in, developed to aid in visualising the results of the combined dynamic and static analyses. © 2005 IEEE. (7 refs)
Main heading: Static analysis
Controlled terms: Computer software reusability
Uncontrolled terms: Dynamic analysis techniques - Existing systems - Plug-ins - Software reconnaissance - Tool support
Classification Code: 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.

Data Provider: Engineering Village

11. Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey

Koschke, Rainer (1)
Source: *Journal of Software Maintenance and Evolution*, v 15, n 2, p 87-109, March/April 2003; **ISSN:** 1532060X; **DOI:** 10.1002/smr.270; **Publisher:** John Wiley and Sons Ltd
Author affiliation: (1) Institut fur Softwaretechnologie, Universität Stuttgart, Breitwiesenstrasse 20-22, 70565 Stuttgart, Germany
Abstract: Software visualization is concerned with the static visualization as well as the animation of software artifacts, such as source code, executable programs, and the data they manipulate, and their attributes, such as size, complexity, or dependencies. Software visualization techniques are widely used in the areas of software maintenance, reverse engineering, and re-engineering, where typically large amounts of complex data need to be understood and a high degree of interaction between software engineers and automatic analyses is required. This paper reports the results of a survey on the perspectives of 82 researchers in software maintenance, reverse engineering, and re-engineering on software visualization. It describes to which degree the researchers are involved in software visualization themselves, what is visualized and how, whether animation is frequently used, whether the researchers believe animation is useful at all, which automatic graph layouts are used if at all, whether the layout algorithms have deficiencies, and - last but not least - where the medium-term and long-term research in software visualization should be directed. The results of this survey help to ascertain the current role of software visualization in software engineering from the perspective of researchers in these domains and give hints on future research avenues. (24 refs)
Main heading: Computer software maintenance
Controlled terms: Algorithms - Animation - Automatic testing - Computational complexity - Human computer interaction - Reengineering - Reverse engineering - Visualization
Uncontrolled terms: Automatic analysis - Software visualization
Classification Code: 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 722.2 Computer Peripheral Equipment - 723.1 Computer Programming - 723.5 Computer Applications - 913.3 Quality Assurance and Control
Treatment: General review (GEN)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

12. An Eclipse plug-in for the detection of design pattern instances through static and dynamic analysis

De Lucia, Andrea (1); Deufemia, Vincenzo (1); Gravino, Carmine (1); Risi, Michele (1)
Source: *IEEE International Conference on Software Maintenance, ICSM*, 2010, *Proceedings - 2010 IEEE International Conference on Software Maintenance, ICSM 2010*; **ISBN-13:** 9781424486298; **DOI:** 10.1109/ICSM.2010.5609707; **Article number:** 5609707; **Conference:** 2010 IEEE International Conference on Software Maintenance, ICSM 2010, September 12, 2010 - September 18, 2010; **Sponsor:** IEEE Computer Society; ACI Worldwide; Microsoft; IBM; Alcatel-Lucent; **Publisher:** IEEE Computer Society
Author affiliation: (1) Dipartimento di Matematica e Informatica, Università degli studi di Salerno, Fisciano(SA), Italy
Abstract: The extraction of design pattern information from software systems can provide conspicuous insight to software engineers on the software structure and its internal charactzeristics. In this demonstration we present ePAD, an Eclipse plug-in for recovering design pattern instances from object-oriented source code. The tool is able to recover design pattern instances through a structural analysis performed on a data model extracted from source code, and a behavioral analysis performed through the instrumentation and the monitoring of the software system. ePAD is fully configurable since it allows software engineers to customize the design pattern recovery rules and the layout used for the visualization of the recovered instances. © 2010 IEEE. (22 refs)
Main heading: Computer software maintenance
Controlled terms: Behavioral research - Computer software - Dynamic analysis - Engineers - Mathematical models - Recovery - Reverse engineering - Structural analysis - Visualization
Uncontrolled terms: Behavioral analysis - Configurable - Data models - Design pattern recovery - Design Patterns - Eclipse plug-in - Object oriented - Plug-ins - Software engineers - Software structures - Software systems - Source code analysis - Source codes - Static and dynamic analysis
Classification Code: 921 Mathematics - 912.4 Personnel - 902.1 Engineering Graphics - 971 Social Sciences - 723 Computer Software, Data Handling and Applications - 422.2 Strength of Building Materials : Test Methods - 408.1 Structural Design, General - 531 Metallurgy and Metallography

Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

13. A lightweight visualization of interprocedural data-flow paths for source code reading

Ishio, Takashi (1); Etsuda, Shogo (1); Inoue, Katsuro (1)
Source: *IEEE International Conference on Program Comprehension*, p 37-46, 2012, *2012 20th IEEE International Conference on Program Comprehension, ICPC 2012 - Proceedings*; **ISBN-13:** 9781467312165; **Article number:** 6240506; **Conference:** 2012 20th IEEE International Conference on Program Comprehension, ICPC 2012, June 11, 2012 - June 13, 2012; **Sponsor:** IEEE; IEEE Computer Society; Technical Council on Software Engineering (TCSE); **Publisher:** IEEE Computer Society
Author affiliation: (1) Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka, Japan
Abstract: To understand the behavior of a program, developers must read source code fragments in various modules. For developers investigating data-flow paths among modules, a call graph is too abstract since it does not visualize how parameters of method calls are related to each other. On the other hand, a system dependence graph is too fine-grained to investigate interprocedural data-flow paths. In this research, we propose an intermediate-level of visualization; we visualize interprocedural data-flow paths among method parameters and fields with summarized intraprocedural data-flow paths. We have implemented our visualization as an Eclipse plug-in for Java. The tool comprises a lightweight data-flow analysis and an interactive graph viewer using fractal value to extract a small subgraph of data-flow related to variables specified by a developer. A case study has shown our visualization enabled developers to investigate more data-flow paths in a fixed time slot. In addition, we report our lightweight data-flow analysis can generate precise data-flow paths for 98% of Java methods. © 2012 IEEE. (23 refs)
Main heading: Visualization
Controlled terms: Computer aided language translation - Computer architecture - Static analysis
Uncontrolled terms: Call graphs - Dataflow - Fixed time - Inter-procedural - Java methods - Plug-ins - Program comprehension - Software visualization - Source codes - Subgraphs - System dependence graph
Classification Code: 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

14. Animated visualization of software history using evolution storyboards

Beyer, Dirk (1); Hassan, Ahmed E. (2)
Source: *Proceedings - Working Conference on Reverse Engineering, WCRE*, p 199-208, 2006, *Proceedings - 13th Working Conference on Reverse Engineering, WCRE 2006*; **ISSN:** 10951350; **ISBN-10:** 0769527191, **ISBN-13:** 9780769527192; **DOI:** 10.1109/WCRE.2006.14; **Article number:** 4023990; **Conference:** 13th Working Conference on Reverse Engineering, WCRE 2006, October 23, 2006 - October 27, 2006; **Publisher:** IEEE Computer Society
Author affiliation: (1) EPFL, Switzerland (2) University of Victoria, Canada
Abstract: The understanding of the structure of a software system can be improved by analyzing the system's evolution during development. Visualizations of software history that provide only static views do not capture the dynamic nature of software evolution. We present a new visualization technique, the Evolution Storyboard, which provides dynamic views of the evolution of a software's structure. An evolution storyboard consists of a sequence of animated panels, which highlight the structural changes in the system; one panel for each considered time period. Using storyboards, engineers can spot good design, signs of structural decay, or the spread of cross cutting concerns in the code. We implemented our concepts in a tool, which automatically extracts software dependency graphs from version control repositories and computes storyboards based on panels for different time periods. For applying our approach in practice, we provide a step by step guide that others can follow along the storyboard visualizations, in order to study the evolution of large systems. We have applied our method to several large open source software systems. In this paper, we demonstrate that our method provides additional information (compared to static views) on the ArgoUML project, an open source UML modeling tool. © 2006 IEEE. (17 refs)
Main heading: Software architecture
Controlled terms: Animation - Information analysis - Static analysis - Unified Modeling Language
Uncontrolled terms: Dependency graphs - Evolution Storyboard - Open source software - Software's structure
Classification Code: 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.5 Computer Applications - 903.1 Information Sources and Analysis
Treatment: Theoretical (THR)

Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

15. Proceedings of the 1998 ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering

Editors: Anon
Source: *ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 1998; **Conference:** Proceedings of the 1998 ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, June 16, 1998 - June 16, 1998; **Sponsor:** ACM; **Publisher:** ACM
Abstract: The proceedings contains 11 papers from the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering: Topics discussed include: experiments with combined analysis for pointer aliasing; OPTVIEW approach to examine optimized code; memory error detection via static point analysis; static slicing of threaded programs; recovering software architecture from multiple source code analysis; composite data flow analysis applied to concurrent programs; software visualization in desert environment; lightweight architecture for program execution monitoring; performance visualization of higher-order programs; and program spectra.
Main heading: Computer aided software engineering
Controlled terms: C (programming language) - Computer architecture - Computer software maintenance - Computer software selection and evaluation - Data flow analysis - Data storage equipment - Data structures - Interfaces (computer) - Large scale systems - Legacy systems - Program compilers - Program debugging
Uncontrolled terms: EiRev - Galois connection - Legacy code modularization - Multiple source code analysis - Pointer aliasing - Program profiling - Software package ManSART - Software package OPTVIEW - Static pointer analysis - Static program slicing
Classification Code: 722.1 Data Storage, Equipment and Techniques - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.2 Data Processing and Image Processing - 723.5 Computer Applications
Treatment: General review (GEN) - Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

16. Case study: Visual analytics in software product assessments

Telea, Alexandru (1); Voinea, Lucian (2)
Source: *Proceedings of VISSOFT 2009 - 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, p 65-72, 2009, *Proceedings of VISSOFT 2009 - 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*; **ISBN-13:** 9781424450251; **DOI:** 10.1109/VISSOF.2009.5336417; **Article number:** 5336417; **Conference:** VISSOFT 2009 - 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, September 25, 2009 - September 25, 2009; **Sponsor:** IEEE Computer Society; IEEE Computer Society Technical; Council on Software Engineering (TCSE); **Publisher:** IEEE Computer Society
Author affiliation: (1) Institute for Math. and Computer Science, University of Groningen, Netherlands (2) SolidSource BV, Eindhoven, Netherlands
Abstract: We present how a combination of static source code analysis, repository analysis, and visualization techniques has been used to effectively get and communicate insight in the development and project management problems of a large industrial code base. This study is an example of how visual analytics can be effectively applied to answer maintenance questions and support decision making in the software industry. We comment on the relevant findings during the study both in terms of used technique and applied methodology and outline the favorable factors that were essential in making this type of assessment successful within tight time and budget constraints. ©2009 IEEE. (18 refs)
Main heading: Project management
Controlled terms: Computer software - Computer software maintenance - Visualization
Uncontrolled terms: Budget constraint - Industrial codes - Software industry - Software products - Static sources - Visual analytics - Visualization technique
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics - 912.2 Management
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

17. Visual exploration of function call graphs for feature location in complex software systems

Bohnet, Johannes (1); Döllner, Jürgen (1)
Source: *Proceedings - SOFTVIS 06: ACM Symposium on Software Visualization*, p 95-104, 2006, *Proceedings - SOFTVIS 06: ACM Symposium on Software Visualization*; **ISBN-10:** 1595934642, **ISBN-13:** 9781595934642; **DOI:** 10.1145/1148493.1148508; **Conference:** SOFTVIS 06: ACM Symposium on Software Visualization, September 4, 2006 - September 5, 2006; **Sponsor:** ACM SIG on Computer Graphics and Interactive Techniques, SIGGRAPH; ACM Special Interest Group on Software Engineering, SIGSOFT; ACM Special Interest Group on Computer-Human Interaction, SIGCHI; ACM Special Interest Group on Programming Languages, SIGPLAN; **Publisher:** Association for Computing Machinery
Author affiliation: (1) University of Potsdam, Hasso-Plattner-Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
Abstract: Maintenance, reengineering, and refactoring processes of software systems are typically driven and organized in terms of features. Feature change requests need to be translated into changes in source code, which is a highly cost intensive and time consuming task when complex legacy software systems are concerned; their documentation is likely to be outdated and incomplete. In this paper, we propose a prototype tool that supports users in locating and understanding feature implementation in large (>1 MLOC) C/C++ systems. A combination of static and dynamic analysis allows extracting of the function call graph during feature execution and interpreting it within the static architecture of the system. An interactive multi-view visualization enables users to explore that graph. An effective 2 1/2D visualization provides various visual cues that facilitate finding those paths in the function call graph that are essential for understanding feature functionality. Additionally to source code aspects, the dynamic metric of function execution times is exploited, which gives significant hints to feature-implementing functions. Furthermore, information on functions is extended by architectural aspects, thereby supporting users in remaining oriented during their analysis and exploration task as they can give priority to selected architectural components and thereby hide insignificant function calls. © 2006 by the Association for Computing Machinery, Inc. (32 refs)
Main heading: Computer software
Controlled terms: Computer architecture - Dynamic analysis - Graph theory - Object oriented programming - Program documentation - Reverse engineering
Uncontrolled terms: Complex software systems - Feature analysis - Feature location - Program comprehensions - Software visualization - Source codes - Static architecture - Time consuming tasks
Classification Code: 422.2 Strength of Building Materials : Test Methods - 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 921.4 Combinatorial Mathematics, Includes Graph Theory, Set Theory
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

18. Static analysis of programs with graphical user interface

Staiger, Stefan (1)
Source: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, p 252-261, 2007, *Proceedings - CSMR 2007: 11th European Conference on Software Maintenance and Reengineering - Software Evolution in Complex Software Intensive Systems*; **ISSN:** 15345351; **DOI:** 10.1109/CSMR.2007.44; **Article number:** 4145043; **Conference:** CSMR 2007: 11th European Conference on Software Maintenance and Reengineering, March 21, 2007 - March 23, 2007; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) Institute of Software Technology, University of Stuttgart
Abstract: We describe a new approach for statically analyzing programs which have a graphical user interface (GUI). Our analysis detects the parts of the program which belong to the GUI, it detects widgets and hierarchies they form, and it shows the event handlers connected to events of those widgets. Besides supporting general program understanding, we show that this also supports control-flow analysis, architecture recovery, migration to GUI builders and mapping the visual appearance of the program to source code arte-facts. Our tests indicate that the static analysis we propose is fast and useful. © 2007 IEEE. (23 refs)
Main heading: Computer programming
Controlled terms: Graphical user interfaces - Software architecture - Static analysis
Uncontrolled terms: Control flow analysis - Source code
Classification Code: 722.2 Computer Peripheral Equipment - 723.1 Computer Programming - 723.5 Computer Applications
Treatment: Theoretical (THR)
Database: Compendex

Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

19. Detecting security vulnerabilities with software architecture analysis tools

Karppinen, Kaarina (1); Lindvall, Mikael (2); Yonkwa, Lyly (2)
Source: *2008 IEEE International Conference on Software Testing Verification and Validation Workshop, ICSTW'08*, p 262-268, 2008, *2008 IEEE International Conference on Software Testing Verification and Validation Workshop, ICSTW'08*; **ISBN-10:** 0769533884, **ISBN-13:** 9780769533889; **DOI:** 10.1109/ICSTW.2008.14; **Article number:** 4567018; **Conference:** 2008 IEEE International Conference on Software Testing Verification and Validation Workshop, ICSTW'08, April 9, 2008 - April 11, 2008; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) VTT Technical Research Centre of Finland (2) FC-MD Fraunhofer Center for Experimental Software Engineering Maryland
Abstract: Hidden functionality in software is a big problem, because we cannot be sure that the software does not contain malicious code. We conducted an experiment where we studied the relationship between architecture constructs, dynamic behavior and security vulnerabilities. We also studied to what extent architecture analysis tools can assist in detecting security vulnerabilities that are caused by architecture violations. Using the tool, we were able to capture the dynamic pattern of a user breaking in to the system using the back door. Based on the dynamic information in combination with the static information, we obtained a good picture of the "visual image" of the back door. Such "visual images" can be used to detect vulnerabilities and ultimately help to design software architectures that meet their security requirements. © 2008 IEEE. (9 refs)
Main heading: Software architecture
Controlled terms: Architecture - Codes (symbols) - Computer software selection and evaluation - Doors - Software design - Software testing - Verification
Uncontrolled terms: Architecture analysis - Design softwares - Dynamic behaviors - Dynamic information - International conferences - Malicious code - Security requirements - Security vulnerabilities - Software architecture analysis - Static information - Verification and validation - Visual imaging
Classification Code: 402 Buildings and Towers - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.2 Data Processing and Image Processing - 723.5 Computer Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

20. Characterising, explaining, and exploiting the approximate nature of static analysis through animation

Binkley, David (1); Harman, Mark (2); Krinke, Jens (3)
Source: *Proceedings - Sixth IEEE International Workshop on Source Code Analysis and Manipulation, SCAM 2006*, p 43-52, 2006, *Proceedings - Sixth IEEE International Workshop on Source Code Analysis and Manipulation, SCAM 2006*; **ISBN-10:** 0769523536, **ISBN-13:** 9780769523538; **DOI:** 10.1109/SCAM.2006.7; **Article number:** 4026854; **Conference:** 6th IEEE International Workshop on Source Code Analysis and Manipulation, SCAM 2006, September 27, 2006 - September 29, 2006; **Sponsor:** IEEE Computer Society Technical Council on Software Engineering; King's College London; Analysis, Slicing and Transformation Research Network (ASTReNet); RainCode; Villanova University; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) Loyola College, Baltimore, MD 21210-2699, United States (2) King's College London, Strand, London WC2R 2LS, United Kingdom (3) FernUniversität in Hagen, 58084 Hagen, Germany
Abstract: This paper addresses the question: "How can animated visualisation be used to express interesting properties of static analysis?" The particular focus is upon static dependence analysis, but the approach adopted in the paper is applicable to other forms of static analysis. The challenge is twofold. First, there is the inherent difficultly of using animation, which is inherently dynamic, as a representation of static analysis, which is not. The paper shows one way in which this apparent contradiction can be overcome. Second, there is the harder challenge of ensuring that the animations so-produced correspond to features of genuine interest in the source code that are hard to visualize without animation. To address these two challenges the paper shows how properties of static dependence analysis can be formulated in a manner suitable for animated visualisation. These formulations of dependence have been implemented and the results used to provide dependence visualisations of the structure of a set of C programs. All animations described in the paper are also viewable on-line. © 2006 IEEE. (44 refs)
Main heading: Computer programming languages
Controlled terms: Animation - Approximation theory - Static analysis
Uncontrolled terms: Source code - Static dependence analysis

Classification Code: 723.1.1 Computer Programming Languages - 723.5 Computer Applications - 921.6 Numerical Methods
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

21. Projecting code changes onto execution traces to support localization of recently introduced bugs

Bohnet, Johannes (1); Voigt, Stefan (1); Döllner, Jürgen (1)
Source: *Proceedings of the ACM Symposium on Applied Computing*, p 438-442, 2009, *24th Annual ACM Symposium on Applied Computing, SAC 2009*; **ISBN-13:** 9781605581668; **DOI:** 10.1145/1529282.1529378; **Conference:** 24th Annual ACM Symposium on Applied Computing, SAC 2009, March 8, 2009 - March 12, 2009; **Sponsor:** ACM SIGAPP; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Hasso-Plattner-Institute, University of Potsdam, Germany
Abstract: Working collaboratively on complex software systems often leads to situations where a developer enhances or extends system functionality, thereby however, introducing bugs. At best the unintentional changes are caught immediately by regression tests. Often however, the bugs are detected days or weeks later by other developers noticing strange system behavior while working on different parts of the system. Then it is a highly time-consuming task to trace back this behavior change to code changes in the past. In this paper we propose a technique for identifying the recently introduced change that is responsible for the unexpected behavior. The key idea is to combine dynamic, static, and code change information on the system to reduce the possibly great amount of code modifications to those that may affect the system while running its faulty behavior. After having applied this massive automated filtering step, developers receive support in semi-automatically identifying the root cause change by means of a trace exploration frontend. Within multiple synchronized views, developers explore when, how and why modified code locations are executed. The technique is implemented within a prototypical analysis tool that copes with large (> MLOC) C/C++ software systems. We demonstrate the approach by means of industrial case studies. Copyright 2009 ACM. (14 refs)
Main heading: Program debugging
Controlled terms: Computer science - Computer software - Dynamic analysis - Visualization
Uncontrolled terms: Analysis tools - Behavior change - Code changes - Code modifications - Complex software systems - Execution trace - Fault localization - Industrial case study - Regression tests - Root cause - Software systems - Software visualization - System behaviors - System functionality - Time-consuming tasks - Trace exploration
Classification Code: 422.2 Strength of Building Materials : Test Methods - 721 Computer Circuits and Logic Elements - 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 902.1 Engineering Graphics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

22. Behavioral pattern identification through visual language parsing and code instrumentation

De Lucia, Andrea (1); Deufemia, Vincenzo (1); Gravino, Carmine (1); Risi, Michele (1)
Source: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, p 99-108, 2009, *Proceedings - 13th European Conference on Software Maintenance and Reengineering, CSMR 2009*; **ISSN:** 15345351; **ISBN-13:** 9780769535890; **DOI:** 10.1109/CSMR.2009.29; **Article number:** 4812743; **Conference:** 13th European Conference on Software Maintenance and Reengineering, CSMR 2009, March 24, 2009 - March 27, 2009; **Sponsor:** Reengineering Forum, REF; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) Dipartimento di Matematica e Informatica, Università di Salerno, 84084 Fisciano, SA, Italy
Abstract: In this paper we present a new technique able to recover behavioral design pattern instances which combines static analysis, based on visual language parsing, with dynamic analysis, based on source code instrumentation. In particular, the dynamic analysis is performed through the automatic instrumentation of the method calls involved in the candidate pattern instances identified during static analysis. The results obtained from a program monitoring activity are matched against the definitions of the pattern behaviors expressed in terms of monitoring grammars. We also present and discuss the results of a case study on JHotDraw 5.1 software library performed to assess the retrieval effectiveness of the proposed approach. © 2009 IEEE. (32 refs)
Main heading: Dynamic analysis
Controlled terms: Computer software - Computer software maintenance - Instruments - Linguistics - Static analysis

Uncontrolled terms: Behavioral patterns - Candidate patterns - Code instrumentation - Design Pattern - Program monitoring - Retrieval effectiveness - Software libraries - Source Code Instrumentation - Visual language parsing
Classification Code: 944 Moisture, Pressure and Temperature, and Radiation Measuring Instruments - 943 Mechanical and Miscellaneous Measuring Instruments - 942 Electric and Electronic Measuring Instruments - 941 Acoustical and Optical Measuring Instruments - 903.2 Information Dissemination - 723.5 Computer Applications - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 422.2 Strength of Building Materials : Test Methods
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

23. Analyzing Java software by combining metrics and program visualization

Systa, Tarja (1); Yu, Ping (1); Muller, Hausi (1)
Source: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, p 199-208, 2000; **Conference:** The 4th European Conference on Software Maintenance and Reegineering - CSMR 2000, February 29, 2000 - March 3, 2000; **Sponsor:** Reengineering Forum; IEEE Computer Society/TCSE; University of Zuric; IT-ieee; **Publisher:** Institute of Electrical and Electronics Engineers Computer Society
Author affiliation: (1) Tampere Univ of Technology, Tampere, Finland
Abstract: Shimba, a prototype reverse engineering environment, has been built to support the understanding of Java software. Shimba uses Rigi and SCED to analyze, visualize, and explore the static and dynamic aspects, respectively, of the subject system. The static software artifacts and their dependencies are extracted from Java byte code and viewed as directed graphs using the Rigi reverse engineering environment. The static dependency graphs of a subject system can be annotated with attributes, such as software quality measures, and then be analyzed and visualized using scripts through the end-user programmable interface. Shimba has recently been extended with the Chidamber and Kemerer suite of object-oriented metrics. The metrics measure properties of the classes, the inheritance hierarchy, and the interaction among classes of a subject system. Since Shimba is primarily intended for the analysis and exploration of Java software, the metrics have been tailored to measure properties of software components written in Java. We show how these metrics can be applied in the context of understanding software systems using a reverse engineering environment. The static dependency graphs of the system under investigation are decorated with measures obtained by applying the object-oriented metrics to selected software components. Shimba provides tools to examine these measures, to find software artifacts that have values that are in a given range, and to detect correlations among different measures. The object-oriented analysis of the subject Java system can be investigated further by exporting the measures to a spreadsheet. (25 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Computer software maintenance - Interfaces (computer) - Java programming language - Object oriented programming - Reverse engineering
Uncontrolled terms: Object oriented metrics - Software package Java
Classification Code: 722.2 Computer Peripheral Equipment - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.1.1 Computer Programming Languages
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

24. MOQA; unlocking the potential of compositional static average-case analysis

Schellekens, M.P. (1)
Source: *Journal of Logic and Algebraic Programming*, v 79, n 1, p 61-83, January 2010; **ISSN:** 15678326; **DOI:** 10.1016/j.jlap.2009.02.006; **Publisher:** Elsevier Inc.
Author affiliation: (1) University College Cork, Department of Computer Science, Centre for Efficiency-Oriented Languages (CEOL), Ireland
Abstract: Compositionality is the "golden key" to static analysis and plays a central role in static worst-case time analysis. We show that compositionality, combined with the capacity for tracking data distributions, unlocks a useful novel technique for average-case analysis. The applicability of the technique has been demonstrated via the static average-case analysis tool DISTRI. The tool automatically extracts average-case time from source code of programs implemented in the novel programming language MOQA1MOdular Quantitative Analysis.1. MOQA enables the prediction of the average number of basic steps performed in a computation, paving the way for static analysis of complexity measures such as average time or average power use. MOQA has as a unique feature a guaranteed average-case timing compositionality. The compositionality property brings a strong advantage for the programmer.

The capacity to combine parts of code, where the average-time is simply the sum of the times of the parts, is a very helpful advantage in static analysis, something which is not available in current languages. Moreover, re-use is a key factor in the MOQA approach: once the average time is determined for a piece of code, then this time will hold in any context. Hence it can be re-used and the timing impact is always the same. Compositionality also improves precision of static average-case analysis, supporting the determination of accurate estimates on the average number of basic operations of MOQA programs. The MOQA "language" essentially consists of a suite of data-structuring operations together with conditionals, for-loops and recursion. As such MOQA can be incorporated in any traditional programming language, importing all of its benefits in a familiar context2MOQA is implemented at CEOL in Java 5.0 as MOQA-java.2. Compositionality for average-case is subtle and one may easily be tempted to conclude that compositionality "comes for free". For genuine compositional reasoning however, one needs to be able to track data and their distribution throughout computations; a non-trivial problem. The lack of an efficient method to track distributions has plagued all prior static average-case analysis approaches. We show how MOQA enables the finitary representation and tracking of the distribution of data states throughout computations. This enables one to unlock the true potential of compositional reasoning. Links with reversible computing are discussed. The highly visual aspect of this novel and unified approach to the Analysis of Algorithms also has a pedagogical advantage, providing students with useful insights in the nature of algorithms and their analysis. © 2009 Elsevier Inc. All rights reserved. (58 refs)
Main heading: Java programming language
Controlled terms: Algorithms - C (programming language) - Computational efficiency - Computer software - Linguistics - Query languages - Static analysis - Time measurement
Uncontrolled terms: Average-case analysis - Compositionality - Randomness preservation - Static power - Static power analysis - Static timing analysis
Classification Code: 921 Mathematics - 903.2 Information Dissemination - 723.5 Computer Applications - 943.3 Special Purpose Instruments - 723.3 Database Systems - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 723.1.1 Computer Programming Languages
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

25. Detecting defects with an interactive code review tool based on visualisation and machine learning

Axelsson, Stefan (1); Baca, Dejan (1); Feldt, Robert (1); Sidlauskas, Darius (1); Kacan, Denis (1)
Source: *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering, SEKE 2009*, p 412-417, 2009, *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering, SEKE 2009*; **ISBN-10:** 1891706241, **ISBN-13:** 9781891706240; **Conference:** 21st International Conference on Software Engineering and Knowledge Engineering, SEKE 2009, July 1, 2009 - July 3, 2009; **Sponsor:** Knowledge Systems Institute Graduate School; **Publisher:** Unavailable
Author affiliation: (1) Blekinge Institute of Technology, Sweden
Abstract: Code review is often suggested as a means of improving code quality. Since humans are poor at repetitive tasks, some form of tool support is valuable. To that end we developed a prototype tool to illustrate the novel idea of applying machine learning (based on Normalised Compression Distance) to the problem of static analysis of source code. Since this tool learns by example, it is trivially programmer adaptable. As machine learning algorithms are notoriously difficult to understand operationally (they are opaque) we applied information visualisation to the results of the learner. In order to validate the approach we applied the prototype to source code from the open-source project Samba and from an industrial, telecom software system. Our results showed that the tool did indeed correctly find and classify problematic sections of code based on training examples. (17 refs)
Main heading: Learning algorithms
Controlled terms: Codes (symbols) - Knowledge engineering - Learning systems - Software engineering - Static analysis - Visualization
Uncontrolled terms: Code quality - Code review - Machine learning algorithms - Machine-learning - Open source projects - Prototype tools - Repetitive task - Software systems - Source codes - TeleCOM - Tool support - Training example - Visualisation
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

26. Visualization of C++ template metaprograms

Borók-Nagy, Zoltán (1); Májer, Viktor (1); Mihalicza, József (1); Pataki, Norbert (1); Porkoláb, Zoltán (1)

Source: *Proceedings - 10th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2010*, p 167-176, 2010, *Proceedings - 10th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2010*; **ISBN-13:** 9780769541785; **DOI:** 10.1109/SCAM.2010.16; **Article number:** 5601850; **Conference:** 10th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2010, September 12, 2010 - September 13, 2010; **Sponsor:** IEEE Computer Society; **Publisher:** Association for Computing Machinery

Author affiliation: (1) Dept. of Programming Languages and Compilers, Eötvös Loránd University, Faculty of Informatics, Pazmany Peter Setany 1/C, H-1117 Budapest, Hungary

Abstract: Template metaprograms have become an essential part of today's C++ programs: with proper template definitions we can force the C++ compiler to execute algorithms at compilation time. Among the application areas of template metaprograms are the expression templates, static interface checking, code optimization with adaptation, language em- bedding and active libraries. Despite all of its already proven benefits and numerous successful applications there are surprisingly few tools for creating, supporting, and ana- lyzing C++ template metaprograms. As metaprograms are executed at compilation time they are even harder to under- stand. In this paper we present a code visualization tool, which is utilizing Templight, our previously developed C++ template metaprogram debugger. Using the tool it is pos- sible to visualize the instantiation chain of C++ templates and follow the execution of metaprograms. Various presen- tation layers, filtering of template instances and step-by- step replay of the instantiations are supported. Our tool can help to test, optimize, maintain C++ template metapro- grams, and can enhance their acceptance in the software industry. © 2010 IEEE. (28 refs)

Main heading: Computer software

Controlled terms: Optimization - Visualization

Uncontrolled terms: Application area - C++ templates - Code optimization - Code visualization - Debuggers - Expression templates - Metaprograms - Software industry - Step-by-step

Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics - 921.5 Optimization Techniques

Database: Compendex

Compilation and indexing terms, Copyright 2013 Elsevier Inc.

Data Provider: Engineering Village

27. Computation and visualization of cause-effect paths

Dubey, Alpana (1); Murthy, Pvr (2)

Source: *2013 8th International Workshop on Automation of Software Test, AST 2013 - Proceedings*, p 139-145, 2013, *2013 8th International Workshop on Automation of Software Test, AST 2013 - Proceedings*; **ISBN-13:** 9781467361613; **DOI:** 10.1109/IWAST.2013.6595805; **Article number:** 6595805; **Conference:** 2013 8th International Workshop on Automation of Software Test, AST 2013, May 18, 2013 - May 19, 2013; **Publisher:** IEEE Computer Society

Author affiliation: (1) Software Development Improvement Program ABB Ltd., Bangalore, India (2) Corporate Research Technologies Siemens, Bangalore, India

Abstract: Static analyzers detect possible run-time errors at compile-time and often employ data-flow analysis techniques to infer properties of programs. Usually, dataflow analysis tools report possible errors with line numbers in source code and leave the task of locating root causes of errors. This paper proposes a technique to aid developers in locating the root causes of statically identified run-time errors with the help of cause-effect paths. A cause effect path terminates at an erroneous statement and originates at the statement which is responsible for the error. We propose modifications to the classic data-flow analysis algorithm to compute cause-effect paths. We discuss different visualization modes in which cause-effect paths can be displayed. As a case study, we implemented a null pointer analyzer, with the additional capability of cause-effect path computation, using the Microsoft Phoenix framework. In addition, we propose a methodology to automatically generate an analyzer which computes cause-effect paths using a framework such as Microsoft Phoenix. © 2013 IEEE. (17 refs)

Main heading: Static analysis

Controlled terms: Computer aided language translation - Computer debugging - Errors - Flow visualization - Software testing - Visualization

Uncontrolled terms: Cause-effect - Compile time - MicroSoft - Path computation - Root cause - Run-time errors - Source codes - Static analyzers

Classification Code: 631.1 Fluid Flow, General - 722 Computer Systems and Equipment - 723.5 Computer Applications - 731 Automatic Control Principles and Applications - 902.1 Engineering Graphics - 921 Mathematics

Database: Compendex

Compilation and indexing terms, Copyright 2013 Elsevier Inc.

Data Provider: Engineering Village

28. Heapviz: Interactive heap visualization for program understanding and debugging

Kelley, Sean (1); Aftandilian, Edward (1); Gramazio, Connor (1); Ricci, Nathan (1); Su, Sara L. (1); Guyer, Samuel Z. (1)
Source: *Information Visualization*, v 12, n 2, p 163-177, April 2013, *Software Visualization*; **ISSN:** 14738716, **E-ISSN:** 14738724; **DOI:** 10.1177/1473871612438786; **Publisher:** SAGE Publications Ltd
Author affiliation: (1) Department of Computer Science, Tufts University, 161 College Ave., Medford, MA 02155, United States
Abstract: Understanding the data structures in a program is crucial to understanding how the program works, or why it does not work. Inspecting the code that implements the data structures, however, is an arduous task and often fails to yield insights into the global organization of a program's data. Inspecting the actual contents of the heap solves these problems but presents a significant challenge of its own: finding an effective way to present the enormous number of objects it contains. In this paper w e present Heapviz, a tool for visualizing and exploring snapshots of the heap obtained from a running Java program. Unlike existing tools, such as traditional debuggers, Heapviz presents a global view of the program state as a graph, together with powerful interactive capabilities for navigating it. Our tool employs several key techniques that help manage the scale of the data. First, we reduce the size and complexity of the graph by using algorithms inspired by static shape analysis to aggregate the nodes that make up a data structure. Second, we implement a powerful visualization component whose interactive interface provides extensive support for exploring the graph. The user can search for objects based on type, connectivity, and field values; group objects; and color or hide and show each group. The user may also inspect individual objects to see their field values and neighbors in the graph. These interactive abilities help the user manage the complexity of these huge graphs. By applying Heapviz to both constructed and real-world examples, we show that it provides programmers with a powerful and intuitive tool for exploring program behavior. © The Author(s) 2012. (28 refs)
Main heading: Data structures
Controlled terms: Computer software - Inspection - Java programming language - Visualization
Uncontrolled terms: Force-directed layout - Graph visualization - Interactive visualizations - Program visualization - Software visualization
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics - 913.3.1 Inspection
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

29. Maintaining a COTS integrated solution - are traditional static analysis techniques sufficient for this new programming methodology?

Cherinka, R. (1); Overstreet, C.M. (1); Ricci, J. (1)
Source: *Conference on Software Maintenance*, p 160-169, 1998; **Conference:** Proceedings of the 1998 IEEE International Conference on Software Maintenance, ICSM, November 16, 1998 - November 20, 1998; **Sponsor:** IEEE; **Publisher:** IEEE
Author affiliation: (1) MITRE Corp, Hampton, United States
Abstract: As integrating commercial off-the-shelf (COTS) products into new homogeneous systems replaces 'traditional' software development approaches, software maintenance problems persist. This approach builds new solutions via 'glue code' using visual languages, which tie together client-based office products, server-based 'BackOffice' products and web-based services/applications. The resulting collection of distributed object-oriented components are glued together by attaching code snippets written in a visual language to other components and controls, such as a command button on a form. A majority of the code in such an application is pre-generated and self-contained in the individual components being reused and, as a result, is typically difficult to understand and maintain. Our experience shows that, while these approaches actually exacerbate some maintenance problems, such as the introduction of dead code, traditional static analysis techniques may still facilitate common maintenance activities. This work reports on the use of data flow techniques on several medium-sized COTS integrated solutions that have become difficult to maintain. We found that by exploiting semantic information, traditional techniques can be augmented to handle some of the unique maintenance issues of component-based software. (18 refs)
Main heading: Software engineering
Controlled terms: Client server computer systems - Computer programming - Computer programming languages - Computer software maintenance - Data flow analysis - Object oriented programming - World Wide Web
Uncontrolled terms: Commercial off the shelf products - Program understanding - Static analysis techniques
Classification Code: 722.4 Digital Computers and Systems - 723.1 Computer Programming - 723.1.1 Computer Programming Languages
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.

Data Provider: Engineering Village

30. Constellation visualization: Augmenting program dependence with dynamic information

Deng, Fang (1); DiGiuseppe, Nicholas (1); Jones, James A. (1)
Source: *Proceedings of VISSOFT 2011 - 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2011, *Proceedings of VISSOFT 2011 - 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis*; **ISBN-13:** 9781457708237; **DOI:** 10.1109/VISSOF.2011.6069453; **Article number:** 6069453; **Conference:** 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2011, September 29, 2011 - September 30, 2011; **Sponsor:** IEEE Computer Society; IEEE Comput. Soc. Tech. Counc. Softw. Eng. (TCSE); **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Informatics, University of California, Irvine, Irvine, CA 92617-3440, United States
Abstract: This paper presents a scalable, statement-level visualization that shows related code in a way that supports human interpretation of clustering and context. The visualization is applicable to many software-engineering tasks through the utilization and visualization of problem-specific meta-data. The visualization models statement-level code relations from a system-dependence- graph model of the program being visualized. Dynamic, run-time information is used to augment the static program model to further enable visual cluster identification and interpretation. In addition, we performed a user study of our visualization on an example program domain. The results of the study show that our new visualization successfully revealed relevant context to the programmer participants. © 2011 IEEE. (15 refs)
Main heading: Data visualization
Controlled terms: Codes (symbols) - Visualization
Uncontrolled terms: Dynamic information - Graph model - Identification and Interpretation - Program dependence - Run-time information - Static program - User study - Visualization models
Classification Code: 723.2 Data Processing and Image Processing - 902.1 Engineering Graphics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

31. Fast analysis of source code in C and C++

Savitskii, V.O. (1); Sidorov, D.V. (1)
Source: *Programming and Computer Software*, v 39, n 1, p 49-55, January 2013; **ISSN:** 03617688; **DOI:** 10.1134/S0361768813010064; **Publisher:** Maik Nauka-Interperiodica Publishing
Author affiliation: (1) Institute for System Programming, Russian Academy of Sciences, ul. Solzhenitsyna 25, Moscow, 109004, Russia
Abstract: Static analysis is a popular tool for detecting the vulnerabilities that cannot be found by means of ordinary testing. The main problem in the development of static analyzers is their low speed. Methods for accelerating such analyzers are described, which include incremental analysis, lazy analysis, and header file caching. These methods make it possible to considerably accelerate the detection of defects and to integrate the static analysis tools in the development environment. As a result, defects in a file edited in the Visual Studio development environment can be detected in 0.5 s or faster, which means that they can be practically detected after each keystroke. Therefore, critical vulnerabilities can be detected and corrected at the stage of coding. © Pleiades Publishing, Ltd., 2013. (8 refs)
Main heading: Static analysis
Controlled terms: Defects
Uncontrolled terms: Detection of defects - Development environment - Header files - Incremental analysis - Low speed - Source codes - Static analyzers - Visual studios
Classification Code: 423 Non Mechanical Properties and Tests of Building Materials - 723.1 Computer Programming - 951 Materials Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

32. Answering common questions about code

LaToza, Thomas D. (1)
Source: *Proceedings - International Conference on Software Engineering*, p 983-986, 2008, *ICSE'08 Companion - Companion Material of the 30th International Conference on Software Engineering*; **ISSN:** 02705257; **ISBN-13:** 9781605580791; **Conference:** 30th International Conference on Software Engineering, ICSE'08, May 10, 2008 - May 18, 2008; **Sponsor:** Special Interest Group on Software Engineering (ACM SIGSOFT); IEEE Computer Society (IEEE CSE); **Publisher:** IEEE Computer Society

Author affiliation: (1) Institute for Software Research, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, United States
Abstract: Difficulties understanding update paths while understanding code cause developers to waste time and insert bugs. A detailed investigation of these difficulties suggests that a wide variety of problems could be addressed by more easily answering questions about update paths that existing tools do not answer. We are designing a feasible update path static analysis to compute these paths and a visualization for asking questions and displaying results. In addition to grounding the questions we answer and tailoring the program analysis in data, we will also evaluate the usefulness of our tool using lab and field studies. (11 refs)
Main heading: Static analysis
Controlled terms: Navigation - Software engineering - Visualization
Uncontrolled terms: Callgraph - Code navigation - Empirical study - Feasible paths - Program comprehension - Science of design
Classification Code: 716.3 Radio Systems and Equipment - 723.1 Computer Programming - 902.1 Engineering Graphics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

33. Tackling software navigation issues of the Smalltalk IDE

Röthlisberger, David (1); Nierstrasz, Oscar (1); Bergel, Alexandre (2); Ducasse, Stéphane (3)
Source: *Proceedings of the International Workshop on Smalltalk Technologies 2009, IWST'09 - ESUG 2009 Smalltalk Joint Event*, p 58-67, 2009, *Proceedings of the International Workshop on Smalltalk Technologies 2009, IWST'09 - ESUG 2009 Smalltalk Joint Event*; **ISBN-13:** 9781605588995; **DOI:** 10.1145/1735935.1735945; **Conference:** International Workshop on Smalltalk Technologies 2009, IWST'09, August 31, 2009 - August 31, 2009; **Sponsor:** ACM Special Interest Group on Programming Languages (SIGPLAN); ESUG; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Software Composition Group, University of Bern, Switzerland (2) Computer Science Department (DCC), University of Chile, Chile (3) INRIA-Lille Nord Europe, France
Abstract: The IDE used in most Smalltalk dialects, including Pharo, Squeak and Cincom Smalltalk, did not evolve significantly over the last years, if not to say decades. For other languages, for instance Java, the available IDEs made tremendous progress as Eclipse and Net-Beans illustrate. While the Smalltalk IDE served as an exemplar for many years, other IDEs caught up or even overtook the erstwhile leader in terms of feature-richness, usability and code navigation facilities. In this paper we first analyze the difficulty of software navigation in the Smalltalk IDE and second illustrate with concrete examples the features we added to the Smalltalk IDE to fill the gap to modern IDEs and to provide novel, improved means to navigate source space. We show that thanks to the agility and dynamics of Smalltalk, we are able to extend and enhance with reasonable effort the Smalltalk IDE to better support software navigation, program comprehension, and software maintenance in general. One such support is the integration of dynamic information into the static source views we are familiar with. Other means include easing the access to static information (for instance by better arranging important packages) or helping developers locating artifacts of interest. Copyright 2009 ACM. (16 refs)
Main heading: Computer software maintenance
Controlled terms: Computer software - Integrodifferential equations - Java programming language - Navigation - Visualization
Uncontrolled terms: Code navigation - Development environment - Dynamic information - Program comprehension - Smalltalk - Software analysis - Software maintenance - source code navigation - Source codes - Source space - Static information - Static sources
Classification Code: 902.1 Engineering Graphics - 723.1.1 Computer Programming Languages - 723 Computer Software, Data Handling and Applications - 921.2 Calculus - 716.3 Radio Systems and Equipment - 434.4 Waterway Navigation - 431.5 Air Navigation and Traffic Control - 655.1 Spacecraft, General
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

34. MAGISTER: Quality assurance of Magic applications for software developers and end users

Nagy, Csaba (1); Vidács, László (1); Ferenc, Rudolf (1); Gyimóthy, Tibor (1); Kocsis, Ferenc (2); Kovács, István (2)
Source: *IEEE International Conference on Software Maintenance, ICSM*, 2010, *Proceedings - 2010 IEEE International Conference on Software Maintenance, ICSM 2010*; **ISBN-13:** 9781424486298; **DOI:** 10.1109/ICSM.2010.5609550;

Article number: 5609550; **Conference:** 2010 IEEE International Conference on Software Maintenance, ICSM 2010, September 12, 2010 - September 18, 2010; **Sponsor:** IEEE Computer Society; ACI Worldwide; Microsoft; IBM; Alcatel-Lucent; **Publisher:** IEEE Computer Society
Author affiliation: (1) University of Szeged, Department of Software Engineering, Research Group on Artificial Intelligence, Hungary (2) SZEGED Software Zrt., Hungary
Abstract: Nowadays there are many tools and methods available for source code quality assurance based on static analysis, but most of these tools focus on traditional software development techniques with 3GL languages. Besides procedural languages, 4GL programming languages such as Magic 4GL and Progress are widely used for application development. All these languages lie outside the main scope of analysis techniques. In this paper we present MAGISTER, which is a quality assurance framework for applications being developed in Magic, a 4GL application development solution created by Magic Software Enterprises. MAGISTER extracts data using static analysis methods from applications being developed in different versions of Magic (v5-9 and uniPaaS). The extracted data (including metrics, rule violations and dependency relations) is presented to the user via a GUI so it can be queried and visualized for further analysis. It helps software developers, architects and managers through the full development cycle by performing continuous code scans and measurements. © 2010 IEEE. (8 refs)
Main heading: Quality control
Controlled terms: Computer software maintenance - Computer software selection and evaluation - Quality assurance - Reverse engineering - Software design - Static analysis
Uncontrolled terms: Analysis techniques - Application development - Dependency relation - Development cycle - End users - Magic 4GL - Metrics - Procedural languages - Programming language - Rule violation - Software developer - Software development techniques - Software enterprise - Source codes - Static analysis method - Tools and methods
Classification Code: 723 Computer Software, Data Handling and Applications - 913.3 Quality Assurance and Control
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

35. Support for static concept location with sv3D

Xie, Xinrong (1); Poshyvanyk, Denys (1); Marcus, Andrian (1)
Source: *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, p 102-107, 2005, *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*; **ISBN-10:** 0780395409, **ISBN-13:** 9780780395404; **DOI:** 10.1109/VISSOF.2005.1684315; **Article number:** 1684315; **Conference:** 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2005, September 25, 2005 - September 25, 2005; **Sponsor:** IEEE Computer Society; **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computer Science, Wayne State University, Detroit, MI 48202, United States
Abstract: The paper presents a new visualization approach to support static concept location in source code. The approach is realized through the combination of two existing tools: IRiSS, which is an information retrieval based tool that support source code searching and browsing; and sv3D, which is a software visualization front end Both tools are integrated into MS Visual Studio NET. The motivation behind the approach, the definition of the visual mappings, and usage examples are also presented in the paper, together with an outline of future and related work. © 2005 IEEE. (16 refs)
Main heading: Tools
Controlled terms: Visualization
Uncontrolled terms: Concept locations - Related works - Software visualization - Source codes - Visual mapping - Visual studios
Classification Code: 603 Machine Tools - 605 Small Tools and Hardware - 902.1 Engineering Graphics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

36. Approach to static prediction and visual analysis of program execution time

Sun, Chang-Ai (1); Jin, Mao-Zhong (1); Liu, Chao (1); Jin, Ruo-Ming (1)
Source: *Ruan Jian Xue Bao/Journal of Software*, v 14, n 1, p 68-75, January 2003; **Language:** Chinese; **ISSN:** 10009825; **Publisher:** Chinese Academy of Sciences
Author affiliation: (1) Dept. of Comp. Sci. and Eng., Beijing Univ. of Aero. and Astron., Beijing 100083, China
Abstract: An important issue of real-time software development is to analyze and predict the execution time of real-time software. A kind of visual prediction and analysis framework of the execution time of real-time software based on program flowchart is proposed. The key issues of implementing the framework are discussed in detail, including

creating the mapping between intermediate code segment and statement line of source code, retrieving the time of any given program segment from the perspective of CPU cycles of goal machine instruct, calculating CPU cycles of statement lines of source code, point-to-point WCETC (worst case execution time calculated) analysis algorithm based on program flowchart, and transforming CPU cycle into physical time. Based on the framework, a practical tool has been developed to predicate and analyze visually the program execution time. Finally, conclusion and comparison between the work in this paper and others is given. (12 refs)
Main heading: Software engineering
Controlled terms: Algorithms - Flowcharting - High level languages
Uncontrolled terms: Program execution time evaluation - Real time software - Software testing
Classification Code: 723 Computer Software, Data Handling and Applications
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

37. Proceedings of the 2002 ACM SIGPLAN-SIGSOFT workshop on program analysis for software tools and engineering

Source: *ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 2002;
Conference: Proceedings of the 4th 2002 ACM SIGPLAN SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, November 18, 2002 - November 19, 2002; **Sponsor:** Association for Computing and Machinery; Special Interest Group om Programming Languages; Special Interest Group on Software Engineering;
Publisher: Association for Computing Machinery
Abstract: The proceedings contains 10 papers from the conference on the Proceedings of the 2002 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. The topics discussed include: monitoring deployed software using software tomography; recompilation for debugging support in JIT-compiler; instruction-level reverse execution for debugging; selective path profiling and combining static and dynamic data in code visualization.
Main heading: Computer aided software engineering
Controlled terms: Algorithms - Binary codes - Computer software selection and evaluation - Computerized tomography - Data compression - Information theory - Java programming language - Motion planning - Program compilers - Program debugging
Uncontrolled terms: Data definition languages - Debugging aids - EiRev - Path profiling - Reverse execution - Software tomography
Classification Code: 716.1 Information Theory and Signal Processing - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.2 Data Processing and Image Processing - 723.4 Artificial Intelligence - 723.5 Computer Applications
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

38. Seesoft--A tool for visualizing line oriented software statistics

Eick, Stephen G. ; Steffen, Joseph L. ; Sumner Jr., Eric E.
Source: *IEEE Transactions on Software Engineering*, v 18, n 11, p 957-968, Nov 1992; **ISSN:** 00985589; **DOI:** 10.1109/32.177365
Abstract: The Seesoft software visualization system allows one to analyze up to 50,000 lines of code simultaneously by mapping each line of code into a thin row. The color of each row indicates a statistic of interest, e.g., red rows are those most recently changed, and blue are those least recently changed. Seesoft displays data derived from a variety of sources, such as version control systems that track the age, programmer, and purpose of the code (e.g., control ISDN lamps, fix bug in call forwarding); static analyses, (e.g., locations where functions are called); and dynamic analyses (e.g., profiling). By means of direct manipulation and high interaction graphics, the user can manipulate this reduced representation of the code in order to find interesting patterns. Further insight is obtained by using additional windows to display the actual code. Potential applications for Seesoft include discovery, project management, code tuning, and analysis of development methodologies. (20 refs)
Main heading: Software engineering
Controlled terms: Color computer graphics - Computer vision - Statistical methods
Uncontrolled terms: Software visualization systems
Classification Code: 723.5 Computer Applications - 922.2 Mathematical Statistics

Treatment: Applications (APP) - Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

39. Experimental and FE analysis of quasi-static bending of foam-filled structures

Kinoshita, Shigeaki (1); Lu, Guoxing (2); Ruan, Dong (1); Beynon, John (1)
Source: *SAE Technical Papers*, p 44-54, 2010; **DOI:** 10.4271/2010-01-0020; **Conference:** SAE 2010 World Congress and Exhibition, April 13, 2010 - April 13, 2010; **Publisher:** SAE International
Author affiliation: (1) Faculty of Engineering and Industrial Sciences, Swinburne University of Technology, John Street, Hawthorn, VIC 3122, Australia (2) Nanyang Technological Univ., Singapore
Abstract: Three-point bending under quasi-static loading was carried out on empty and partially foam-filled tubes. ALPORASAˆ aluminium foam was used as an insert with aluminium alloy tube in the experiment. The experiment was modelled using the finite element (FE) code software package LS-DYNAAˆ. The Deshpande-Fleck constitutive model was implemented in the simulation to model the foam. Its material parameters were calibrated against experimental data. The material parameters for the tube were also derived experimentally. Experimental data showed increase in specific energy absorption of approximately 17% whilst the lowest generator of the tube was found to increase by approximately 14% with inclusion of partially filling foam. The FE simulations compared well with the experiments, with the load-displacement and LG-indenter displacement agreeing within approximately 15%. Visual inspection of the foam showed local densification around the point of indentation. Similar features were seen in the FE simulation with high strains and large deformation present near the point of indentation. The results thus indicated a localised effect in energy absorption of the foam in the scenario being studied. Copyright © 2010 SAE International. (31 refs)
Main heading: Computer simulation
Controlled terms: Bending (forming) - Energy absorption - Experiments - Structural analysis - Tubes (components)
Uncontrolled terms: Aluminium alloy tube - Aluminium foam - Experimental data - FE analysis - FE-simulation - Finite element codes - Foam-filled tubes - High strains - Large deformations - Load displacements - Material parameter - Quasi-static - Quasi-static loading - Specific energy absorption - Three point bending - Visual inspection
Classification Code: 408.1 Structural Design, General - 535.2 Metal Forming - 616.1 Heat Exchange Equipment and Components - 723.5 Computer Applications - 901.3 Engineering Research - 931.3 Atomic and Molecular Physics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

40. Experimental and FE analysis of quasi-static bending of foam-filled structures

Kinoshita, Shigeaki (1); Lu, Guoxing (2); Ruan, Dong (1); Beynon, John (1)
Source: *SAE International Journal of Materials and Manufacturing*, v 3, n 1, p 44-54, August 2010; **ISSN:** 19463979, **E-ISSN:** 19463987; **DOI:** 10.4271/2010-01-0020; **Publisher:** SAE International
Author affiliation: (1) Faculty of Engineering and Industrial Sciences, Swinburne University of Technology, John Street, Hawthorn, VIC.3122, Australia (2) Nanyang Technological University, Singapore
Abstract: Three-point bending under quasi-static loading was carried out on empty and partially foam-filled tubes. ALPORAS® aluminium foam was used as an insert with aluminium alloy tube in the experiment. The experiment was modelled using the finite element (FE) code software package LS-DYNA®. The Deshpande-Fleck constitutive model was implemented in the simulation to model the foam. Its material parameters were calibrated against experimental data. The material parameters for the tube were also derived experimentally. Experimental data showed increase in specific energy absorption of approximately 17% whilst the lowest generator of the tube was found to increase by approximately 14% with inclusion of partially filling foam. The FE simulations compared well with the experiments, with the load displacement and LG-indenter displacement agreeing within approximately 15%. Visual inspection of the foam showed local densification around the point of indentation. Similar features were seen in the FE simulation with high strains and large deformation present near the point of indentation. The results thus indicated a localised effect in energy absorption of the foam in the scenario being studied. © 2010 SAE International. (31 refs)
Main heading: Computer simulation
Controlled terms: Absorption - Aluminum - Bending (forming) - Energy absorption - Experiments - Iron alloys - Structural analysis - Tubes (components) - Vickers hardness testing
Uncontrolled terms: Aluminium alloy tube - Aluminium foam - Experimental data - FE analysis - FE-simulation - Finite element codes - High strains - Indenters - Large deformations - Load displacements - LS-DYNA - Material parameter - Quasi-static - Quasi-static loading - Specific energy absorption - Three point bending - Visual inspection
Classification Code: 931.2 Physical Properties of Gases, Liquids and Solids - 901.3 Engineering Research - 723.5 Computer Applications - 616.1 Heat Exchange Equipment and Components - 931.3 Atomic and Molecular Physics

- 545.3 Steel - 535.2 Metal Forming - 422.2 Strength of Building Materials : Test Methods - 408.1 Structural Design, General - 541.1 Aluminum
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

41. Compiler Construction: 17th International Conference, CC 2008 - Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Proceedings

Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 4959 LNCS, 2008, *Compiler Construction - 17th International Conference, CC 2008 - Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3540787909, **ISBN-13:** 9783540787907; **Conference:** 17th International Conference on Compiler Construction, CC 2008, March 29, 2008 - April 6, 2008; **Publisher:** Springer Verlag
Abstract: The proceedings contain 19 papers. The topics discussed include: design choices in a compiler course or how to make undergraduates love formal notation; improved memory-access analysis for x86 executables; a system for generating static analyzers for machine instructions; IDE dataflow analysis in the presence of large object-oriented libraries; an adaptive strategy for inline substitution; automatic transformation of bit-level C code to support multiple equivalent data layouts; control flow emulation on tiled SIMD architectures; generating SIMD vectorized permutations; how to do a million watchpoints: efficient debugging using dynamic instrumentation; compiler-guaranteed safety in code-copying virtual machines; hardware JIT compilation for off-the-shelf dynamically reconfigurable FPGAs; visualization of program dependence graphs; and on the relative completeness of bytecode analysis versus source code analysis.
Main heading: Program compilers
Controlled terms: Access control - Adaptive control systems - Codes (standards) - Codes (symbols) - Computer debugging - Concurrency control - Control system analysis - Curricula - Data flow analysis - Data storage equipment - Data visualization - Digital libraries - Machine design - Military data processing - Program debugging - Security of data - Static analysis
Uncontrolled terms: Adaptive strategies - Automatic transformations - Byte codes - Compiler construction - Conferences (Chemical industry) - Control flows - Data layouts - Data-flow analysis - Dynamic instrumentation - European - Executables - International conferences - Just in time (JIT) compilation - Machine instructions - Object oriented libraries - Program dependence graph (PDG) - Re-configurable - SIMD architectures - Source code analysis - Static analyzers - Virtual machine (VM)
Classification Code: 901.2 Education - 731.1 Control Systems - 723.5 Computer Applications - 723.3 Database Systems - 902.2 Codes and Standards - 723.2 Data Processing and Image Processing - 723 Computer Software, Data Handling and Applications - 722.1 Data Storage, Equipment and Techniques - 601 Mechanical Design - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

42. Heapviz: Interactive heap visualization for program understanding and debugging

Aftandilian, Edward E. (1); Kelley, Sean (1); Gramazio, Connor (1); Ricci, Nathan (1); Su, Sara L. (1); Guyer, Samuel Z. (1)
Source: *Proceedings of the ACM Conference on Computer and Communications Security*, p 53-62, 2010, *SOFTVIS'10 - Proceedings of the 2010 International Symposium on Software Visualization, Co-located with VisWeek 2010*; **ISSN:** 15437221; **ISBN-13:** 9781450304948; **DOI:** 10.1145/1879211.1879222; **Conference:** 2010 5th International Symposium on Software Visualization, SOFTVIS'10, Co-located with VisWeek 2010, October 25, 2010 - October 26, 2010; **Sponsor:** ACM SIGCHI; ACM SIGPLAN; ACM SIGGRAPH; ACM SIGSOFT; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Department of Computer Science, Tufts University, United States
Abstract: Understanding the data structures in a program is crucial to understanding how the program works, or why it doesn't work. Inspecting the code that implements the data structures, however, is an arduous task and often fails to yield insights into the global organization of a program's data. Inspecting the actual contents of the heap solves these problems but presents a significant challenge of its own: finding an effective way to present the enormous number of objects it contains. In this paper we present Heapviz, a tool for visualizing and exploring snapshots of the heap obtained from a running Java program. Unlike existing tools, such as traditional debuggers, Heapviz presents a global view of the program state as a graph, together with powerful interactive capabilities for navigating it. Our tool employs several key techniques that help manage the scale of the data. First, we reduce the size and complexity of the graph

by using algorithms inspired by static shape analysis to aggregate the nodes that make up a data structure. Second, we introduce a dominator-based layout scheme that emphasizes hierarchical containment and ownership relations. Finally, the interactive interface allows the user to expand and contract regions of the heap to modulate data structure detail, inspect individual objects and field values, and search for objects based on type or connectivity. By applying Heapviz to both constructed and real-world examples, we show that Heapviz provides programmers with a powerful and intuitive tool for exploring program behavior. Copyright 2010 ACM. (32 refs)
Main heading: Program debugging
Controlled terms: Computer software - Data structures - Inspection - Java programming language - Visualization
Uncontrolled terms: Debugging - Graphs - Interactive visualizations - Program understanding - Software visualization
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics - 913.3.1 Inspection
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

43. Monitoring compliance of a software system with its high-level design models

Sefika, Mohlalefi (1); Sane, Aamod (1); Campbell, Roy H. (1)
Source: *Proceedings - International Conference on Software Engineering*, p 387-396, 1995; **ISSN:** 02705257; **Conference:** Proceedings of the 1996 18th International Conference on Software Engineering, March 25, 1996 - March 29, 1996; **Sponsor:** IEEE; **Publisher:** IEEE
Author affiliation: (1) Univ of Illinois at Urbana-Champaign, Urbana, United States
Abstract: As a complex software system evolves, its implementation tends to diverge from the intended or documented design models. Such undesirable deviation makes the system hard to understand, modify, and maintain. This paper presents a hybrid computer-assisted approach for confirming that the implementation of a system maintains its expected design models and rules. Our approach closely integrates logic-based static analysis and dynamic visualization, providing multiple code views and perspectives. We show that the hybrid technique helps determine design-implementation congruence at various levels of abstraction: concrete rules like coding guidelines, architectural models like design patterns[7] or connectors[26], and subjective design principles like low coupling and high cohesion. The utility of our approach has been demonstrated in the development of µChoices, a new multimedia operating system which inherits many design decisions and guidelines learned from experience in the construction and maintenance of its predecessor, Choices. (26 refs)
Main heading: Computer aided software engineering
Controlled terms: Computer aided design - Computer architecture - Computer operating systems - Computer programming - Computer simulation - Computer software - High level languages - Hybrid computers
Uncontrolled terms: Complex software system - Dynamic visualization - High level design models - Logic based static analysis
Classification Code: 722.5 Analog and Hybrid Computers - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.5 Computer Applications
Treatment: Applications (APP)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

44. Kinetic parameters evaluation of PWRs using static cell and core calculation codes

Jahanbin, Ali (1); Malmir, Hessam (1)
Source: *Annals of Nuclear Energy*, v 41, p 110-114, March 2012; **ISSN:** 03064549; **DOI:** 10.1016/j.anucene.2011.11.018; **Publisher:** Elsevier Ltd
Author affiliation: (1) Department of Energy Engineering, Sharif University of Technology, Azadi Street, Tehran, Iran
Abstract: In this paper, evaluation of the kinetic parameters (effective delayed neutron fraction and prompt neutron lifetime) in PWRs, using static cell and core calculation codes, is reported. A new software has been developed to link the WIMS, BORGES and CITATION codes in Visual C# computer programming language. Using the WIMS cell calculation code, multigroup microscopic cross-sections and number densities of different materials can be generated in a binary file. By the use of BORGES code, these binary-form cross-sections and number densities are converted to a format readable by the CITATION core calculation code, by which the kinetic parameters can be finally obtained. This software is used for calculation of the kinetic parameters in a typical VVER-1000 and NOK Beznau reactor. The ratios (#eff)i(#eff) core, which are the important input data for the reactivity accident analysis, are also calculated. Benchmarking of the results against the final safety analysis report (FSAR) of the aforementioned reactors shows very good agreements with these published documents. © 2011 Published by Elsevier Ltd. (12 refs)

Main heading: Neutrons
Controlled terms: Cells - Codes (symbols) - Computer programming - Kinetic parameters - Reactor cores
Uncontrolled terms: Accident analysis - Binary files - Calculation code - Core calculations - Effective delayed neutron fraction - Input datas - Multi-group - Number density - Power reactor - Prompt-neutron lifetime - Safety analysis reports - Visual C
Classification Code: 461.2 Biological Materials and Tissue Engineering - 621.1.1 Fission Reactor Equipment and Components - 723.1 Computer Programming - 723.2 Data Processing and Image Processing - 931 Classical Physics; Quantum Theory; Relativity - 931.3 Atomic and Molecular Physics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

45. T-Morph: Revealing buggy behaviors of TinyOS applications via rule mining and visualization

Zhou, Yangfan (1, 2); Chen, Xinyu (1); Lyu, Michael R. (2, 3); Liu, Jiangchuan (4)
Source: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE 2012*, 2012, *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE 2012*; **ISBN-13:** 9781450316149; **DOI:** 10.1145/2393596.2393615; **Conference:** 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, FSE 2012, November 11, 2012 - November 16, 2012; **Sponsor:** Assoc. Comput. Mach., Spec. Interest; Group Softw. Eng. (ACM SIGSOFT);
Publisher: Association for Computing Machinery
Author affiliation: (1) Shenzhen Research Institute, Chinese Univ. of Hong Kong, Shenzhen, China (2) Dept. of Computer Sci. and Eng., Chinese Univ. of Hong Kong, Hong Kong, Hong Kong (3) School of Computers, National Univ. of Defense Technology, Changsha, China (4) School of Computing Science, Simon Fraser Univ., Burnaby, BC, Canada
Abstract: TinyOS applications for Wireless Sensor Networks (WSNs) typically run in a complicated concurrency model. It is difficult for developers to precisely predict the dynamic execution process of a TinyOS application by its static source codes. Such a conceptual gap frequently incurs software bugs, due to unexpected system behaviors caused by unknown execution patterns. This paper presents T-Morph (TinyOS application tomography), a novel tool to mine, visualize, and verify the execution patterns of TinyOS applications. T-Morph abstracts the dynamic execution process of a TinyOS application into simple, structured application behavior models, which well reflect how the static source codes are executed. Furthermore, T-Morph visualizes them in a user-friendly manner. Therefore, WSN developers can readily see if their source codes run as intended by simply verifying the correctness of the models. Finally, the verified models allow T-Morph to automatically check the application behaviors during a long-term testing execution. The suggested model violations can unveil potential bugs and direct developers to suspicious locations in the source codes. We have implemented T-Morph and applied it to verify a series of representative real-life TinyOS applications and find several bugs, including a new bug in the latest release of TinyOS. It shows T-Morph can provide substantial help to verify TinyOS applications. © 2012 ACM. (35 refs)
Main heading: Program debugging
Controlled terms: Computer programming languages - Dynamic analysis - Sensor networks - Software engineering - Wireless sensor networks
Uncontrolled terms: Application behavior models - Dynamic execution - Rule mining - Software bug - Source codes - Static sources - System behaviors - TinyOS - Wireless sensor network (WSNs)
Classification Code: 422.2 Strength of Building Materials : Test Methods - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 732 Control Devices
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

46. Cognitive design elements to support the construction of a mental model during software exploration

Storey, M.-A.D. (1); Fracchia, F.D. (1); Muller, H.A. (1)
Source: *Journal of Systems and Software*, v 44, n 3, p 171-185, Jan 1 1999; **ISSN:** 01641212; **DOI:** 10.1016/S0164-1212(98)10055-9; **Publisher:** Elsevier Science Inc
Author affiliation: (1) Simon Fraser Univ, Burnaby, Canada
Abstract: The scope of software visualization tools which exist for the navigation, analysis and presentation of software information varies widely. One class of tools, which we refer to as Software exploration tools, provides graphical representations of static software structures linked to textual views of the program source code and

documentation. This paper describes a hierarchy of cognitive issues which should be considered during the design of a software exploration tool. The hierarchy of cognitive design elements is derived through the examination of program comprehension cognitive models. Examples of how existing tools address each of these issues are provided. In addition, this paper demonstrates how these cognitive design elements may be applied to the design of an effective interface for software exploration. (72 refs)
Main heading: Computer aided software engineering
Controlled terms: Cognitive systems - Graphical user interfaces - Hierarchical systems - Utility programs - Visualization
Uncontrolled terms: Cognitive design elements - Software exploration tools
Classification Code: 722.2 Computer Peripheral Equipment - 723.1 Computer Programming - 723.4 Artificial Intelligence - 723.5 Computer Applications
Treatment: Literature review (LIT) - Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

47. Concurrent object-oriented programming: A visualization challenge

Widjaja, Hendra (1); Oudshoorn, Michael J. (1)
Source: *Proceedings of SPIE - The International Society for Optical Engineering*, v 3017, p 310-321, 1997, *Visual Data Exploration and Analysis IV*; **ISSN:** 0277786X; **DOI:** 10.1117/12.270311; **Conference:** Visual Data Exploration and Analysis IV, February 12, 1997 - February 12, 1997; **Publisher:** SPIE
Author affiliation: (1) Department of Computer Science, University of Adelaide, SA 5005, Australia
Abstract: Understanding and subsequently fine-tuning concurrent object-oriented programs may be difficult. To alleviate this situation, program visualization can be used. This research focuses on the question of what and how such visualization can be done for concurrent object-oriented systems. Furthermore, in the absence of language support, to what extent such visualization can be realized. To investigate these issues, Visor++, a tool for visualizing CC++ programs, is developed. This research proposes that both static and dynamic views of programs are important. However, many languages, including CC++, do not provide adequate support for program visualization. CC++, in particular, provides rudimentary support which is available only in the low-level run-time system. Therefore, proper support must be facilitated by Visor++. To make such support portable and maintainable, it is provided at the CC++ source-code level. Although some information, such as changes in variable and data structure values, cannot be easily captured by such an approach, experiments with Visor++ have shown that the information obtained can be of valuable assistance for understanding and fine-tuning programs. ©2004 Copyright SPIE - The International Society for Optical Engineering. (21 refs)
Main heading: Object oriented programming
Controlled terms: Concurrency control - Data structures - Linguistics - Programming theory - Query languages - Tuning - Visualization
Uncontrolled terms: CC++ - Concurrent - Object-oriented - Program visualisation - Software visualisation - Visualisation
Classification Code: 902.1 Engineering Graphics - 744.1 Lasers, General - 731.3 Specific Variables Control - 723.3 Database Systems - 723.2 Data Processing and Image Processing - 903.2 Information Dissemination - 723.1.1 Computer Programming Languages - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 716.4 Television Systems and Equipment - 716.3 Radio Systems and Equipment - 713 Electronic Circuits - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

48. An interactive change impact analysis based on an architectural reflexion model approach

Kim, Tae-Hyung (1); Kim, Kimun (1); Kim, Woomok (1)
Source: *Proceedings - International Computer Software and Applications Conference*, p 297-302, 2010, *Proceedings - 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2010*; **ISSN:** 07303157; **ISBN-13:** 9780769540856; **DOI:** 10.1109/COMPSAC.2010.37; **Article number:** 5676272; **Conference:** 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2010, July 19, 2010 - July 23, 2020; **Sponsor:** IEEE; IEEE Computer Society; **Publisher:** IEEE Computer Society
Author affiliation: (1) Software Engineering Lab., DMC RandD Center, Samsung Electronics, Korea, Republic of

Abstract: To establish the software architecture based on the source code and analyze the impact of its change, we provide an architectural reflexion model that is a simplified version of the reflexion model and an interactive change impact analysis tool. The architectural reflexion model is performed in order to reconstruct the software architecture of a software system based on the static analysis information extracted from its source code. When the architecture of the software system is refined and established, its internal elements affected by the changes can be visualized by the interactive impact analysis tool we implement. The main objective of our approach is to support maintenance of rapidly changing and evolving software systems, usually developed in high-technology companies by means of providing an agile and practical way to reconstruct the software architecture of a large-scale software system and let the user take a snapshot of its internal dependencies on a basis of changes. A case study using an open source project of the large-scale embedded software platform for mobile phone products illustrates how our approach applied and presents its usefulness and effectiveness. © 2010 IEEE. (13 refs)
Main heading: Software architecture
Controlled terms: Computer applications - Computer software maintenance - Embedded software - Maintenance of way - Open systems - Static analysis - Telecommunication equipment
Uncontrolled terms: Change impact analysis - High-technology - Impact analysis - Large-scale software systems - Model approach - Open source projects - Software systems - Source codes
Classification Code: 681 Railway Plant and Structures - 716 Telecommunication; Radar, Radio and Television - 717 Optical Communication - 718 Telephone Systems and Related Technologies; Line Communications - 723 Computer Software, Data Handling and Applications - 723.5 Computer Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

49. K-scope: A Java-based Fortran source code analyzer with graphical user interface for performance improvement

Terai, Masaaki (1); Murai, Hitoshi (1); Minami, Kazuo (1); Yokokawa, Mitsuo (1); Tomiyama, Eiji (2)
Source: *Proceedings of the International Conference on Parallel Processing Workshops*, p 434-443, 2012, *Proceedings - 41st International Conference on Parallel Processing Workshops, ICPPW 2012*; **ISSN:** 15302016; **ISBN-13:** 9780769547954; **DOI:** 10.1109/ICPPW.2012.61; **Article number:** 6337511; **Conference:** 41st International Conference on Parallel Processing Workshops, ICPPW 2012, September 10, 2012 - September 13, 2012; **Sponsor:** Int. Assoc. Comput. Commun. (IACC); **Publisher:** Institute of Electrical and Electronics Engineers Inc.
Author affiliation: (1) RIKEN Advanced Institute for Computational Science, 1-26, Minatojima-minami-machi 7-chome, Chuo-ku, Kobe, Hyogo 650-0047, Japan (2) Research Organization for Information Science and Technology, 3F Kobe KIMEC Center Bldg., 1-5-2, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan
Abstract: Given that scientific computer programs are becoming larger and more complicated, high performance application developers routinely examine the program structure of their source code to improve their performance. We have developed K-scope, a source code analysis tool that can be used to improve code performance. K-scope has graphical user interface that visualizes program structures of Fortran 90 and FORTRAN 77 source code and enables static data-flow analysis. To develop the tool, we adopted the filtered abstract syntax tree (filtered-AST) model with Java to visualize the program structure efficiently. Filtered-AST, which extends the AST in the structured programming model by abstract block structuring, is suitable for visualization program structures. Based on this model, K-scope has been developed as an experimental implementation. It constructs filtered-AST objects from both source and intermediate code generated by the front-end of the XcalableMP compiler. We provide illustrations of the graphical user interface and give detailed examples of the tool applied to an actual application code. © 2012 IEEE. (20 refs)
Main heading: Java programming language
Controlled terms: Computer software - FORTRAN (programming language) - Graphical user interfaces - Program compilers - Structured programming
Uncontrolled terms: Abstract Syntax Trees - Application codes - Code performance - Core performance - filtered-AST - Fortran 77 - Fortran 90 - High performance applications - Performance improvements - Program structures - Source code analysis - Source codes - Static sources - XcalableMP compiler
Classification Code: 722.2 Computer Peripheral Equipment - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

50. Supporting impact analysis by program dependence graph based forward slicing

Korpi, Jaakko (1); Koskinen, Jussi (2)

Source: *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, p 197-202, 2007, *Advances and Innovations in Systems, Computing Sciences and Software Engineering*; **ISBN-13:** 9781402062636; **DOI:** 10.1007/978-1-4020-6264-3_35; **Conference:** 2006 International Conference on Systems, Computing Sciences and Software Engineering, SCSS 2006, Part of the International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, CISSE 2006, December 4, 2006 - December 14, 2006; **Publisher:** Springer Netherlands
Author affiliation: (1) Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland (2) Department of Computer Science and Information Systems, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland
Abstract: Since software must evolve to meet the typically changing requirements, source code modifications can not be avoided. Impact analysis is one of the central and relatively demanding tasks of software maintenance. It is constantly needed while aiming at ensuring the correctness of the made modifications. Due to its importance and challenging nature automated support techniques are required. Theoretically, forward slicing is a very suitable technique for that purpose. Therefore, we have implemented a program dependence graph (PDG) based tool, called GRACE, for it. For example, due to the typical rewritings of Visual Basic programs there is a great need to support their impact analysis. However, there were neither earlier scientific studies on slicing Visual Basic nor reported slicers for it. In case of forward slicing there is a need to perform efficient static slicing revealing all the potential effects of considered source code modifications. Use of PDGs helps in achieving this goal. Therefore, this paper focuses on describing automated PDG-based forward slicing for impact analysis support of Visual Basic programs. GRACE contains a parser, a PDG-generator and all other necessary components to support forward slicing. Our experiences on the application of the PDGbased forward slicing has confirmed the feasibility of the approach in this context. GRACE is also compared to other forward slicing tools. © 2007 Springer. (27 refs)
Main heading: Visual BASIC
Controlled terms: Software engineering
Uncontrolled terms: Automated support - Impact analysis - Potential effects - Program dependence graph - Scientific studies - Source code modification - Static slicing - Visual basic programs
Classification Code: 723.1 Computer Programming - 723.1.1 Computer Programming Languages
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

51. Event-based visualization debugging on concurrent program

Xu, Baowen ; Zuo, Fubin ; Zhou, Xiaoyu ; Shi, Liang ; Zeng, Yi
Source: *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, v 16, p 827-832, 2004, *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*; **ISSN:** 10272658; **Article number:** 439-067; **Conference:** Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems, November 9, 2004 - November 11, 2004; **Sponsor:** Int. Association of Sci. and Technol. for Dev., IASTED; World Modelling and Simulation Forum (WMSF); **Publisher:** Acta Press
Abstract: Traditional cyclic debugging technique cannot be effectively applied to debug concurrent programs because of the non-determinism of programs execution. This paper puts forward a new method to debug concurrent programs based on events, which records the sequence of synchronized events in program execution, makes the execution deterministic and replays the original status of errors according to event records. With this method, therefore, we will be able to implement the one-step debugging at synchronized operation level, and visualize the communication among threads combined with static program analysis. In a word, this method makes the cyclic debugging remain effective in concurrent situations. (13 refs)
Main heading: Program debugging
Controlled terms: Computational complexity - Concurrent engineering - Error analysis - Synchronization - Syntactics - Visualization
Uncontrolled terms: Concurrent programs - Process scheduling - Software testing - Source codes
Classification Code: 921.6 Numerical Methods - 913.6 Product Development; Concurrent Engineering - 903.2 Information Dissemination - 731.1 Control Systems - 723.5 Computer Applications - 723.1 Computer Programming - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

52. SCAM 2007 - Proceedings Seventh IEEE International Working Conference on Source Code Analysis and Manipulation

Editors: Anon
Source: *SCAM 2007 - Proceedings 7th IEEE International Working Conference on Source Code Analysis and Manipulation*, *SCAM 2007 - Proceedings 7th IEEE International Working Conference on Source Code Analysis and Manipulation*, 2007; **ISBN-10:** 0769528805, **ISBN-13:** 9780769528809; **Conference:** 7th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2007, Sep 30 - Oct 1 2007; **Sponsor:** IEEE Computer Society Technical Council on Software Engineering; CEA Laboratoire d'Integration des Systemes et des Technologies; Ecole Polytechnique (France); Software Improvement Group; CREST, King's College London; **Publisher:** Institute of Electrical and Electronics Engineers Computer Society
Abstract: The proceedings contain 23 papers. The topics discussed include: an evaluation of slicing algorithms for concurrent programs; barrier slicing for remote software trusting; statement-level cohesion metrics and their visualization; on temporal path conditions in dependence graphs; towards path-sensitive points-to analysis; extending attribute grammars with collection attributes-valuation and applications; reengineering standard Java runtime systems through dynamic bytecode instrumentation; an integrated crosscutting concern migration strategy and its application to JHOTDRAW; fast approximate matching of programs for protecting libre/open source software by using spatial indexes; finding inputs that reach a target expression; improved static resolution of dynamic class loading in Java; DATES: design analysis tool for enterprise systems; source code composition with the reuseware composition framework; and quality assessment for embedded SQL.
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

53. Active code completion

Omar, Cyrus (1); Yoon, YoungSeok (1); LaToza, Thomas D. (1); Myers, Brad A. (1)
Source: *Proceedings - 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011*, p 261-262, 2011, *Proceedings - 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011*; **ISBN-13:** 9781457712456; **DOI:** 10.1109/VLHCC.2011.6070422; **Article number:** 6070422; **Conference:** 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011, September 18, 2011 - September 22, 2011; **Sponsor:** IEEE; IEEE Computer Society; United States National Science Foundation (NSF); Microsoft Research; National Instruments; **Publisher:** IEEE Computer Society
Author affiliation: (1) School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, United States
Abstract: Software developers today make heavy use of the code completion features available in modern code editors [1]. By navigating and selecting from a floating menu containing the names of variables, fields, methods, types and code snippets, a developer can avoid many common spelling and logic errors, avoid unnecessary keystrokes and explore unfamiliar APIs without leaving the editor window. To ensure that the items featured in this menu are relevant, the editor conducts a static analysis of the surrounding code context. © 2011 IEEE. (4 refs)
Main heading: Static analysis
Controlled terms: Codes (symbols)
Uncontrolled terms: Logic errors - Software developer
Classification Code: 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

54. BSAA: A switching activity analysis and visualisation tool for SoC power optimisation

English, Tom (1); Lok Man, Ka (2); Popovici, Emanuel (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 5953 LNCS, p 216-226, 2010, *Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation - 19th International Workshop, PATMOS 2009, Revised Selected Papers*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3642118011, **ISBN-13:** 9783642118012; **DOI:** 10.1007/978-3-642-11802-9_26; **Conference:** 19th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS 2009, September 9, 2009 - September 11, 2009; **Publisher:** Springer Verlag
Author affiliation: (1) Dept. of Microelectronic Engineering, University College Cork, Ireland (2) Centre for Efficiency-Oriented Languages, University College Cork, Ireland
Abstract: We present Bus Switching Activity Analyser (BSAA), a switching activity analysis and visualisation tool for SoC power optimisation. BSAA reads switching metrics from RTL simulation, reporting the most active buses and hierarchies. Buses with typical address and data bus traffic are identified automatically. The tool can process multiple

simulation runs simultaneously, analysing how switching varies with input data or software code. BSAA complements commercial tools, helping the designer find opportunities to apply power-saving techniques. To illustrate BSAA's powerful features, we analyse switching in an MP3 decoder design using several audio inputs and in a microcontroller running a suite of software tasks. We demonstrate the tool's usefulness by applying it in the power optimisation of a small MPSoC, obtaining on average a 60% reduction in dynamic power across five software tasks and identifying opportunities to reduce static power. © 2010 Springer Berlin Heidelberg. (12 refs)
Main heading: Computer software
Controlled terms: Buses - Design - Modulation - Optimization - Programmable logic controllers - Switching - Technical presentations - Time measurement - Timing circuits - Visualization
Uncontrolled terms: Audio input - Commercial tools - Data bus - Dynamic Power - Input datas - MP3 decoders - Power optimisation - Power-saving - Software codes - Software tasks - Static power - Switching activities - Visualisation
Classification Code: 943.3 Special Purpose Instruments - 723 Computer Software, Data Handling and Applications - 732.1 Control Equipment - 901.2 Education - 902.1 Engineering Graphics - 903.2 Information Dissemination - 921.5 Optimization Techniques - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 717 Optical Communication - 716 Telecommunication; Radar, Radio and Television - 713.4 Pulse Circuits - 663.1 Heavy Duty Motor Vehicles - 408 Structural Design - 718 Telephone Systems and Related Technologies; Line Communications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

55. YARN: Animating software evolution

Hindle, Abram (1); Jiang, Zhen Ming (1); Koleilat, Walid (1); Godfrey, Michael W. (1); Holt, Richard C. (1)
Source: *VISSOFT 2007 - Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, p 129-136, 2007, *VISSOFT 2007 - Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*; **ISBN-10:** 1424406005, **ISBN-13:** 9781424406005; **DOI:** 10.1109/VISSOF.2007.4290711; **Article number:** 4290711; **Conference:** VISSOFT 2007 - 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, June 25, 2007 - June 26, 2007; **Sponsor:** IEEE Computer Society; IEEE Comput. Soc. Tech. Council on Software Engineering (TCSE); IEEE Comput. Soc. Visualization and Graphics Tech. Comm. (VGTC); **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) University of Waterloo, University of Victoria
Abstract: A problem that faces the study of software evolution is how to explore the aggregated and cumulative effects of changes that occur within a software system over time. In this paper we describe an approach to modeling, extracting, and animating the architectural evolution of a software system. We have built a prototype tool called YARN (Yet Another Reverse-engineering Narrative) that implements our approach; YARN mines the source code changes of the target system, and generates YARN "balls" (animations) that a viewer can unravel (watch). The animation is based on a static layout of the modules connected by animated edges that model the changing dependencies. The edges can be weighted by the number of dependencies or the importance of the change. We demonstrate our approach by visualizing the evolution of PostgreSQL DBMS. (24 refs)
Main heading: Agricultural products
Controlled terms: Animation - Computer software - Wool - Yarn
Uncontrolled terms: Cumulative effects - Evolution (CO) - International (CO) - PostgreSQL - Prototype tools - Software Evolution - Software systems - Source code changes - Target systems
Classification Code: 723 Computer Software, Data Handling and Applications - 723.5 Computer Applications - 819.4 Fiber Products - 821.4 Agricultural Products
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

56. Socialization in an open source software community: A socio-technical analysis

Ducheneaut, Nicolas (1)
Source: *Computer Supported Cooperative Work: CSCW: An International Journal*, v 14, n 4, p 323-368, August 2005; **ISSN:** 09259724, **E-ISSN:** 15737551; **DOI:** 10.1007/s10606-005-9000-1; **Publisher:** Kluwer Academic Publishers
Author affiliation: (1) Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, United States
Abstract: Open Source Software (OSS) development is often characterized as a fundamentally new way to develop software. Past analyses and discussions, however, have treated OSS projects and their organization mostly as a static phenomenon. Consequently, we do not know how these communities of software developers are sustained and

reproduced over time through the progressive integration of new members. To shed light on this issue I report on my analyses of socialization in a particular OSS community. In particular, I document the relationships OSS newcomers develop over time with both the social and material aspects of a project. To do so, I combine two mutually informing activities: ethnography and the use of software specially designed to visualize and explore the interacting networks of human and material resources incorporated in the email and code databases of OSS. Socialization in this community is analyzed from two perspectives: as an individual learning process and as a political process. From these analyses it appears that successful participants progressively construct identities as software craftsmen, and that this process is punctuated by specific rites of passage. Successful participants also understand the political nature of software development and progressively enroll a network of human and material allies to support their efforts. I conclude by discussing how these results could inform the design of software to support socialization in OSS projects, as well as practical implications for the future of these projects. © Springer 2005. (78 refs)
Main heading: Software engineering
Controlled terms: Codes (standards) - Computer networks - Database systems - Electronic mail - Learning systems
Uncontrolled terms: Actor-network - Material resources - Open Source - Socialization
Classification Code: 716 Telecommunication; Radar, Radio and Television - 723.1 Computer Programming - 723.3 Database Systems - 723.4 Artificial Intelligence - 723.5 Computer Applications - 902.2 Codes and Standards
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

57. Information visualisation utilising 3D computer game engines case study: A source code comprehension tool

Kot, Blazej (1); Wuensche, Burkhard (1); Grundy, John (1); Hosking, John (1)
Source: *ACM International Conference Proceeding Series*, v 94, p 53-60, 2005, *Proceedings - CHINZ 2005 - Making CHI Natural: 6th International Conference NZ Chapter of the ACM's Special Interest Group on Computer-Human Interaction (SIGCHI-NZ)*; **ISBN-10:** 1595930361, **ISBN-13:** 9781595930361; **DOI:** 10.1145/1073943.1073954;
Conference: 6th International Conference NZ Chapter of the ACM's Special Interest Group on Computer-Human Interaction (SIGCHI-NZ): Making CHI Natural, CHINZ 2005, July 7, 2005 - July 8, 2005; **Sponsor:** The University of Auckland; New Zealand Chapter of ACM SIGCHI; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Department of Computer Science, University of Auckland, New Zealand
Abstract: Information visualisation applications have been facing ever-increasing demands as the amount of available information has increased exponentially. With this, the number and complexity of visualisation tools for analysing and exploring data has also increased dramatically, making development and evolution of these systems difficult. We describe an investigation into reusing technology developed for computer games to create collaborative information visualisation tools. A framework for using game engines for information visualisation is presented together with an analysis of how the capabilities and constraints of a game engine influence the mapping of data into graphical representations and the interaction with it. Based on this research a source code comprehension tool was implemented using the Quake 3 computer game engine. It was found that game engines can be a good basis for an information visualisation tool, provided that the visualisations and interactions required meet certain criteria, mainly that the visualisation can be represented in terms of a limited number of discrete, interactive, and physical entities placed in a static 3-dimensional world of limited size. Copyright 2005 ACM. (31 refs)
Main heading: Human computer interaction
Controlled terms: Computer crime - Computer software - Engines - Interactive computer systems - Knowledge management - Visualization
Uncontrolled terms: 3-dimensional - Collaborative information - Computer game - Computer game engine - Game Engine - Graphical representations - Software visualisation - Source codes - Visualisation
Classification Code: 903.3 Information Retrieval and Use - 902.3 Legal Aspects - 902.1 Engineering Graphics - 723.5 Computer Applications - 723 Computer Software, Data Handling and Applications - 722.4 Digital Computers and Systems - 722.2 Computer Peripheral Equipment - 654.2 Rocket Engines - 617.3 Steam Engines - 612 Engines - 461.4 Ergonomics and Human Factors Engineering
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

58. The SourceGraph program

Miljenovic, Ivan Lazar (1)

Source: *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, p 151-154, 2010, *PEPM'10 - Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*; **ISBN-13:** 9781605587271; **DOI:** 10.1145/1706356.1706384; **Conference:** 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM'10, January 18, 2010 - January 19, 2010; **Sponsor:** ACM SIGPLAN; **Publisher:** Association for Computing Machinery
Author affiliation: (1) School of Mathematics and Physics, University of Queensland, QLD, Australia
Abstract: As software has increased in size and complexity, a range of tools has been developed to assist programmers in analysing the structure of their code. One of the key concepts used for such analysis is the concept of a call graph, which is used to represent which entities in a code base call other entities. However, most tools which use call graphs are limited to either visualisation for documentation purposes (such as Doxygen) or for dynamic analysis to find areas to optimise in the software using profiling tools such as gprof. SourceGraph is a new tool which takes a different approach to software analysis using call graphs, for projects written in Haskell. It creates a static call graph directly from the source code and then uses it to perform static analysis using graph-theoretic techniques with the aim of helping the programmer understand how the different parts of their program interact with each other. Whilst still a work in progress, it can already be used to find possible problems in the code base such as unreachable areas and cycles or cliques in the function calls as well as other useful information. SourceGraph thus provides programmers not only with various ways of visualising their software, but helps them to understand what their code is doing and how to improve it. Copyright © 2010 ACM. (16 refs)
Main heading: Dynamic analysis
Controlled terms: Computer software selection and evaluation - Graph theory - Static analysis - Technical presentations - Visualization - XML
Uncontrolled terms: Call graph - Call graphs - Function calls - Graph-theoretic technique - Haskell - New tools - Profiling tools - Software analysis - Source codes - Visualisation - Work in progress
Classification Code: 921.4 Combinatorial Mathematics, Includes Graph Theory, Set Theory - 912.2 Management - 903.2 Information Dissemination - 902.1 Engineering Graphics - 901.2 Education - 723.5 Computer Applications - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 422.2 Strength of Building Materials : Test Methods
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

59. Proceedings - 29th International Conference on Software Engineering, ICSE 2007

Source: *Proceedings - International Conference on Software Engineering*, 2007, *Proceedings - 29th International Conference on Software Engineering, ICSE 2007*; **ISSN:** 02705257; **ISBN-10:** 0769528287, **ISBN-13:** 9780769528281; **Conference:** 29th International Conference on Software Engineering, ICSE 2007, May 20, 2007 - May 26, 2007; **Sponsor:** IEEE Computer Society Technical Council on Software Engineering; ACM Special Interest Group on Software Engineering, SIGSOFT; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Abstract: The proceedings contain 89 papers. The topics discussed include: sequential circuits for relational analysis; a sound assertion semantics for the dependable systems evolution verifying compiler; behaviour model synthesis from properties and scenarios; feature oriented model driven development: a case study for portlets; feedback-directed random test generation; compatibility and regression testing of COTS-component-based software; very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder; using server pages to unify clones in Web applications: a trade-off analysis; automated inference of pointcuts in aspect-oriented refactoring; a formal framework for automated round-trip software engineering in static aspect weaving and transformations; and identifying feature interactions in multi-language aspect-oriented frameworks.
Main heading: Software engineering
Controlled terms: Computer programming languages - Program compilers - Regression analysis - Semantics - Sequential circuits - Servers
Uncontrolled terms: Behavior model synthesis - Relational analysis - Sound assertion semantics
Classification Code: 721.2 Logic Elements - 722 Computer Systems and Equipment - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 903.2 Information Dissemination - 922.2 Mathematical Statistics
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

60. Maintenance tools

Oman, Paul (1); Novobilski, Andrew (1); Rajlich, Vaclav (1); Harband, Joel (1); McCabe Jr., Thomas (1); Cross, James (1); Vanek, Leonard (1); Davis, Linda (1); Gallagher, Keith (1); Wilde, Norman (1)
Source: *IEEE Software*, v 7, n 3, p 59-65, May 1990; **ISSN:** 07407459; **DOI:** 10.1109/52.55229
Author affiliation: (1) Univ of Idaho, Idaho, ID, USA
Abstract: After a brief overview by P. Oman, eight tools to help the maintenance programmer analyze and understand code are described in separate presentations. All of them are code-visualization tools. However, while all these tools show how a program is structured, they use different means to achieve different ends. The tools covered are: Objective-C Browser; Vifor; Seela; Battle Map; Act; Grasp/Ada; Expert Dataflow and Static Analysis; Surgeon's Assistant; and Dependency Analysis Tool Set.
Main heading: Computer Programming
Controlled terms: Computer Software
Uncontrolled terms: Code-Visualization Tools - Expert Dataflow and Static Analysis - Grasp/Ada Tool - Maintenance Tools - Objective-C Browser Tool - Vifor Tool
Treatment: Applications (APP)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

61. Answering common questions about code

LaToza, Thomas D. (1)
Source: *Proceedings - International Conference on Software Engineering*, p 983-986, 2008, *ICSE'08: Proceedings of the 30th International Conference on Software Engineering 2008*; **ISSN:** 02705257; **ISBN-13:** 9781605580791; **DOI:** 10.1145/1370175.1370218; **Conference:** 30th International Conference on Software Engineering 2008, ICSE'08, May 10, 2008 - May 18, 2008; **Sponsor:** ACM SIGSOFT; IEEE CSE; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) Institute for Software Research, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213
Abstract: Difficulties understanding update paths while understanding code cause developers to waste time and insert bugs. A detailed investigation of these difficulties suggests that a wide variety of problems could be addressed by more easily answering questions about update paths that existing tools do not answer. We are designing a feasible update path static analysis to compute these paths and a visualization for asking questions and displaying results. In addition to grounding the questions we answer and tailoring the program analysis in data, we will also evaluate the usefulness of our tool using lab and field studies. (11 refs)
Main heading: Software engineering
Controlled terms: Navigation
Uncontrolled terms: Callgraph - Code navigation - Empirical study - Feasible paths - Program comprehension - Science of design
Classification Code: 431.5 Air Navigation and Traffic Control - 434.4 Waterway Navigation - 655.1 Spacecraft, General - 716.3 Radio Systems and Equipment - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

62. Proceedings of the 2006 International Workshop on Dynamic Analysis, WODA '06, Co-located with the 28th International Conference on Software Engineering, ICSE 2006

Source: *Proceedings - International Conference on Software Engineering*, 2006, *Proceedings of the 2006 International Workshop on Dynamic Analysis, WODA '06, Co-located with the 28th International Conference on Software Engineering, ICSE 2006*; **ISSN:** 02705257; **ISBN-10:** 1595934006, **ISBN-13:** 9781595934000; **Conference:** 4th International Workshop on Dynamic Analysis, WODA'06, Co-located with the 28th International Conference on Software Engineering, ICSE 2006, May 20, 2006 - May 28, 2006; **Sponsor:** Assoc. Comput. Mach., Spec. Interest; Group Softw. Eng. (ACM SIGSOFT); **Publisher:** IEEE Computer Society
Abstract: The proceedings contain 12 papers. The topics discussed include: isolating relevant component interactions with JINSI; program partitioning - a framework for combining static and dynamic analysis; mining object behavior with ADABU; inferring state-based behavior models; recognizing behavioral patterns at runtime using finite automata; analyzing feature implementation by visual exploration of architecturally-embedded call-graphs; web application characterization through directed requests; a dynamic analysis for revealing object ownership and sharing; dynamic code instrumentation to detect and recover from return address corruption; an empirical study of the strength of information flows in programs; and Grexmk: speeding up scripted builds.

Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

63. Visualizing feature interaction in 3-D

Greevy, Orla (1); Lanza, Michele (2); Wysseier, Christoph (1)
Source: *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, p 114-119, 2005, *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*; **ISBN-10:** 0780395409, **ISBN-13:** 9780780395404; **DOI:** 10.1109/VISSOF.2005.1684317; **Article number:** 1684317; **Conference:** 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2005, September 25, 2005 - September 25, 2005; **Sponsor:** IEEE Computer Society; **Publisher:** IEEE Computer Society
Author affiliation: (1) Software Composition Group, University of Berne, Switzerland (2) Faculty of Informatics, University of Lugano, Switzerland
Abstract: Without a clear understanding of how features of a software system are implemented, a maintenance change in one part of the code may risk adversely affecting other features. Feature implementation and relationships between features are not explicit in the code. To address this problem, we propose an interactive 3D visualization technique based on a combination of static and dynamic analysis which enables the software developer to step through visual representations of execution traces. We visualize dynamic behaviors of execution traces in terms of object creations and interactions and represent this in the context of a static class-hierarchy view of a system. We describe how we apply our approach to a case study to visualize and identify common parts of the code that are active during feature execution. © 2005 IEEE. (20 refs)
Main heading: Three dimensional computer graphics
Uncontrolled terms: Dynamic behaviors - Feature interactions - Interactive 3d visualizations - Object creation - Software developer - Software systems - Static and dynamic analysis - Visual representations
Classification Code: 723.5 Computer Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

64. Slicing droids: Program slicing for smali code

Hoffmann, Johannes (1); Ussath, Martin (1); Holz, Thorsten (1); Spreitzenbarth, Michael (2)
Source: *Proceedings of the ACM Symposium on Applied Computing*, p 1844-1851, 2013, *28th Annual ACM Symposium on Applied Computing, SAC 2013*; **ISBN-13:** 9781450316569; **DOI:** 10.1145/2480362.2480706; **Conference:** 28th Annual ACM Symposium on Applied Computing, SAC 2013, March 18, 2013 - March 22, 2013; **Sponsor:** ACM Special Interest Group on Applied Computing (SIGAPP); ISEC Engenharia; Politecnico de Coimbra; Caixa Geral de Depositos; Institute of Systems and Robotics (ISR), University of Coimbra; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Ruhr-University Bochum, Germany (2) Friedrich-Alexander-University, Erlangen-Nuremberg, Germany
Abstract: The popularity of mobile devices like smartphones and tablets has increased significantly in the last few years with many millions of sold devices. This growth also has its drawbacks: attackers have realized that smartphones are an attractive target and in the last months many different kinds of malicious software (short: malware) for such devices have emerged. This worrisome development has the potential to hamper the prospering ecosystem of mobile devices and the potential for damage is huge. Considering these aspects, it is evident that malicious apps need to be detected early on in order to prevent further distribution and infections. This implies that it is necessary to develop techniques capable of detecting malicious apps in an automated way. In this paper, we present SAAF, a Static Android Analysis Framework for Android apps. SAAF analyzes smali code, a disassembled version of the DEX format used by Android's Java VM implementation. Our goal is to create program slices in order to perform data-flow analyses to backtrack parameters used by a given method. This helps us to identify suspicious code regions in an automated way. Several other analysis techniques such as visualization of control flow graphs or identification of ad-related code are also implemented in SAAF. In this paper, we report on program slicing for Android and present results obtained by using this technique to analyze more than 136,000 benign and about 6,100 malicious apps. Copyright 2013 ACM. (30 refs)
Main heading: Robots
Controlled terms: Computer crime - Mobile devices - Smartphones
Uncontrolled terms: Analysis frameworks - Analysis techniques - Android apps - Control flow graphs - Malicious software - Malwares - Program slice - Program slicing

Classification Code: 716 Telecommunication; Radar, Radio and Television - 718.1 Telephone Systems and Equipment - 723 Computer Software, Data Handling and Applications - 731.5 Robotics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

65. The limited impact of individual developer data on software defect prediction

Bell, Robert M. (1); Ostrand, Thomas J. (1); Weyuker, Elaine J. (1)
Source: *Empirical Software Engineering*, v 18, n 3, p 478-505, June 2013; **ISSN:** 13823256, **E-ISSN:** 15737616; **DOI:** 10.1007/s10664-011-9178-4; **Publisher:** Springer Netherlands
Author affiliation: (1) ATandT Labs Research, 180 Park Avenue, Florham Park, NJ 07932, United States
Abstract: Previous research has provided evidence that a combination of static code metrics and software history metrics can be used to predict with surprising success which files in the next release of a large system will have the largest numbers of defects. In contrast, very little research exists to indicate whether information about individual developers can profitably be used to improve predictions. We investigate whether files in a large system that are modified by an individual developer consistently contain either more or fewer faults than the average of all files in the system. The goal of the investigation is to determine whether information about which particular developer modified a file is able to improve defect predictions. We also extend earlier research evaluating use of counts of the number of developers who modified a file as predictors of the file's future faultiness. We analyze change reports filed for three large systems, each containing 18 releases, with a combined total of nearly 4 million LOC and over 11,000 files. A buggy file ratio is defined for programmers, measuring the proportion of faulty files in Release R out of all files modified by the programmer in Release R-1. We assess the consistency of the buggy file ratio across releases for individual programmers both visually and within the context of a fault prediction model. Buggy file ratios for individual programmers often varied widely across all the releases that they participated in. A prediction model that takes account of the history of faulty files that were changed by individual developers shows improvement over the standard negative binomial model of less than 0.13% according to one measure, and no improvement at all according to another measure. In contrast, augmenting a standard model with counts of cumulative developers changing files in prior releases produced up to a 2% improvement in the percentage of faults detected in the top 20% of predicted faulty files. The cumulative number of developers interacting with a file can be a useful variable for defect prediction. However, the study indicates that adding information to a model about which particular developer modified a file is not likely to improve defect predictions. © 2011 Springer Science+Business Media, LLC. (25 refs)
Main heading: Forecasting
Controlled terms: Defects - Mathematical models - Regression analysis - Research
Uncontrolled terms: Buggy file ratio - Empirical studies - Fault-percentile average - Fault-prone - Regression model - Software fault
Classification Code: 423 Non Mechanical Properties and Tests of Building Materials - 901.3 Engineering Research - 921 Mathematics - 922.2 Mathematical Statistics - 951 Materials Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

66. BinThavro: Towards a useful and fast tool for goodware and malware analysis

Caillat, Benjamin (1); Desnos, Anthony (1); Erra, Robert (1)
Source: *9th European Conference on Information Warfare and Security 2010, ECIW 2010*, p 405-415, 2010, *9th European Conference on Information Warfare and Security 2010, ECIW 2010*; **ISBN-13:** 9781622765355; **Conference:** 9th European Conference on Information Warfare and Security 2010, ECIW 2010, July 1, 2010 - July 2, 2010; **Publisher:** Academic Conferences Limited
Author affiliation: (1) ESIEA, Paris, France
Abstract: We present our tool BinThavro , which helps to solve the following general problem : given two programs, how can we compare them? More precisely, how can we understand the similarities, but also the dissimilarities between both files? The most difficult but the most interesting case seems to be the case of executable (binary) files and this problem has an important application: the malware analysis. A malware is one of the main tools used by information warfare warriors, "bad guys" commonly called "cyberwarriors". He´las, there are so many new malwares that appears quite each day that we need automatic tools to make the analysis faster, and more sure. In the Microsoft Security Intelligence Report it is pointed out that, for the first half of the year 2009, around 116 million malicious samples were detected "in the wild" while this number was around 95 million in the second half of the year 2008. Of course, there does not exist 116 million of dissimilar malwares, a lot of them are clones, similar or quite similar. This proves clearly that the "malware industry" is flourishing, and it is an important arm for the cyberwarriors involded in

the information warfare. But of course, a lot of "new" malwares share large portions of codes with existing and already known malwares (a lot of malwares contains small or large parts of code that has been copied from another). Here, known means analyzed, i.e. we have understood for example what the malware does, how it is programmed, how we can detect him with the help of a static signature in an antivirus software and so on. Why does someone wants to analyze a malware? There are (at least) three reasons: we want to understand how we can be protected against it, with or without a antirus software; or, we want to understand how we can modify to create a new variant (possibly with new functionalities for example); we want to "name" a new malware (see (Gheorgescu 2005)); So, at the first glance, any tool that can be used to analyze a malware can have bad consequences because it will probably be used also to create new malwares. Yes, but this is true for any new language, any new compiler etc. So, beyond the basic idea of searching for a signature of a malware, is there an interest to develop new of better tools for malware or goodware analysis? Yes, for at least two reasons: 1.a new view is emerging the last years: if we have better tools to analyze quickly new malwares that are variants of known malwares, the malwares programmers have to work harder (and so, hopefully, longer) to create new malwares that are difficult to analyze; 2.in the few last years, a new threat has appeared in the tools used for the information warfare: Targeted Malware Attacks, i.e. malwares that are developed to attack a specific target. This is really a serious problem because the reaction of the AV community faced to a new malware depends a lot of the impact of this new malware. And there are so many new malwares that the analyze of new malwares is prioritized, ressources has to be managed in a balance between the importance of the threats and the ability to analyze a lot of files. There are some reasons why this problem is interesting also for goodwares, for example: we want to detect plagiarism or copyright infringement (mostly for goodwares); we want to understand the evolution of a software (both for goodwares and malwares); we want to detect vulnerabilities in an old version by comparing the patched and unpatched versions (mostly for malwares) we want to detect redundancy into a software (mostly for goodwares). The malware analysis problem has clearly close relations with the clone detection problem and so, techniques for comparing goodware files can be used to compare malwares. We can of course also think to the reversed view: new techniques to compare malwares can be used to compare goodwares. For example, the last five years a lot of works has been done about the problem of the dynamic analysis of malwares; we can use these techniques of dynamic analysis to analyze goodwares. Another example is the visualization of a software to help its analyze: if you have such a tool, you can use it for goodwares or malwares. We are interested here to present what problems we have to develop a tool that could help to analyze goodwares and malwares: for goodwares: we will suppose we have a unique file or a few file to compare and analyze; for malwares: we will suppose we have a unknown malware and a large database of (already) known malwares, we want to understand how we can be protected against it (for example we want to ding a signature for AV softwares). We focus mainly on this work on the global malware filtering problem. Let us suppose we have: an unknown malware A, possibly new, this is our "target"; a (large) database of known malwares M={M1,. ..Mn} ; our problem is : how can we choose quickly, from a set of known files {M1,. .. Mn} , the subset of the files the "most similar" to a target A ? We propose to use filtering tactics to select the better files of the malware set M. We propose to use two different but similar tactics, using the Normalized Compression Distance (NCD) for a first filtering tactic to filter the set M and the entropy for a second filtering tactic. We also use the NCD and the entropy at two different levels of granularity using the Control Flow Graphs (CFG) and the Call Graph (CG). With these tools we will define local filtering tactics and we will show how to use them to define our global filtering strategy. The tool BinThavro is not yet available, it is a set of tools, but asap we will likely make it available when we will have a nice GUI. (23 refs)

Main heading: Computer crime
Controlled terms: Cloning - Entropy
Uncontrolled terms: Antivirus softwares - Automatic tools - Call graphs - Clone detection - Compare and analyze - Control flow graphs - Copyright infringement - Fast tool - Filtering problems - Filtering strategies - Information warfare - Large database - Malware analysis - Malware attacks - Malwares - MicroSoft - Normalized compression distance - Static signatures
Classification Code: 461.8.1 Genetic Engineering - 641.1 Thermodynamics - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

67. A programming language that combines the benefits of static and dynamic typing

Ortin, Francisco (1); Garcia, Miguel (1)

Source: *Communications in Computer and Information Science*, v 170, p 72-87, 2011, *Software and Data Technologies - 5th International Conference, ICSOFT 2010, Revised Selected Paper*; **ISSN:** 18650929; **ISBN-13:** 9783642295775; **DOI:** 10.1007/978-3-642-29578-2; **Conference:** 5th International Conference on Software and Data Technologies, ICSOFT 2010, July 22, 2010 - July 24, 2010; **Sponsor:** Inst. for Syst. and Technol. of; Inf. Control and Commun. (INSTICC); **Publisher:** Springer Verlag

Author affiliation: (1) University of Oviedo, Computer Science Department, Calvo Sotelo s/n, 33007, Oviedo, Spain
Abstract: Dynamically typed languages have recently turned out to be suitable for developing specific scenarios where dynamic adaptability or rapid prototyping are important issues. However, statically typed programming languages commonly offer more opportunities for compiler optimizations and earlier type error detection. Due to the benefits of both approaches, some programming languages such as C# 4.0, Boo, Visual Basic or Objective-C provide both static and dynamic typing. We describe the StaDyn programming language that supports both type systems in the very same programming language. The main contribution of StaDyn is that it keeps gathering type information at compile time even over dynamically typed references, obtaining a better runtime performance, earlier type error detection, and an intuitive combination of statically and dynamically typed code. © Springer-Verlag Berlin Heidelberg 2011. (35 refs)
Main heading: Computer programming languages
Controlled terms: Ada (programming language) - Program compilers
Uncontrolled terms: Alias analysis - Duck typing - Separation of concerns - Static typing - Type inferences - Type systems - Union types
Classification Code: 723.1 Computer Programming - 723.1.1 Computer Programming Languages
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

68. eFlowMining: An exception-flow analysis tool for .NET applications

Garcia, Israel (1); Cacho, Nélio (2)
Source: *Proceedings - 5th Latin-American Symposium on Dependable Computing Workshops, LADCW 2011*, p 1-8, 2011, *Proceedings - 5th Latin-American Symposium on Dependable Computing Workshops, LADCW 2011*; **ISBN-13:** 9780769543949; **DOI:** 10.1109/LADCW.2011.18; **Article number:** 5773431; **Conference:** 5th Latin-American Symposium on Dependable Computing Workshops, LADCW 2011, April 25, 2011 - April 29, 2011; **Sponsor:** Sociedade Brasileira de Computacao, SBC; Coordenacao Aperfeicoamento Pessoal Nivel Super., CAPES; Cons. Nac. Desenvolv. Cient. Tecnol., CNPq; Fundacao de Amparo a Pesquisa do Estado de Sao Paulo, FAPESP; Agencia Espacial Brasileira, AEB; **Publisher:** IEEE Computer Society
Author affiliation: (1) Departamento de Engenharia de Produção, Federal University of Rio Grande do Norte, Natal, Brazil (2) Escola de Ciências e Tecnologia, Federal University of Rio Grande do Norte, Natal, Brazil
Abstract: In this paper, we present a exception-flow analysis tool, called eFlowMining, that automates the process of gathering and visualizing exception-handling constructs in multi-programming languages. More specifically, the current version of eFlowMining focuses on code written using the .NET framework to help developers in inspecting applications either to improve them or to understand their exception handling behavior. eFlowMining extracts metrics and information about the exception flows in .NET applications, providing different views of the exception handling. For instance, an Evolution View allows developers to visualize the behavior of multiple metrics over the application history. Use of this tool on five .NET applications demonstrates that the tool can be helpful to support developers building and evolving applications with appropriate error-handling strategies. © 2011 IEEE. (26 refs)
Main heading: Static analysis
Uncontrolled terms: Analysis tools - Error-handling - Exception handling - NET framework - Software evolution
Classification Code: 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

69. Answering control flow questions about code

Toza, Thomas D. La (1)
Source: *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, p 921-922, 2008, *OOPSLA'08: Companion to the 23rd ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*; **DOI:** 10.1145/1449814.1449909; **Conference:** 23rd ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications,OOPSLA 2008, October 19, 2008 - October 23, 2008; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Institute for Software Research, School of Computer Science, Carnegie Mellon University
Abstract: Empirical observations of developers editing code revealed that difficulties following control flow relationships led to poor changes, wasted time, and bugs. I am designing static analysis to compute interprocedural path-sensitive control flow to help developers more quickly and accurately visually answer these common questions about code. (8 refs)
Main heading: Object oriented programming
Controlled terms: Computer systems programming - Data flow analysis - Linguistics

Uncontrolled terms: Control flows - Empirical study - Following controls - Inter procedurals - Program comprehension
Classification Code: 723.1 Computer Programming - 903.2 Information Dissemination
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

70. MeMo - methods of model quality

Hu, Wei (1); Wegener, Joachim (1); Stürmer, Ingo (2); Reicherdt, Robert (3); Salecker, Elke (3); Glesner, Sabine (3)
Source: *Tagungsband - Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII, MBEES 2011*, p 127-132, 2011, *Tagungsband - Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII, MBEES 2011*; **Conference:** Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII - 7th Workshop on Model-Based Development of Embedded Systems, MBEES 2011, February 16, 2011 - February 18, 2011; **Sponsor:** Delta Energy Systems GmbH; Validas AG; **Publisher:** TU Clausthal
Author affiliation: (1) Berner and Mattner Systemtechnik GmbH, Germany (2) Model Engineering Solutions GmbH, Germany (3) Technische Universität Berlin, Germany
Abstract: Model driven development as implemented by the Simulink-Stateflow- TargetLink tool chain facilitates the efficient development of software for embedded processors. But there are only a few automated quality assurance techniques comparable to those known from traditional software development that can be applied in early phases of the development process. This is a serious problem since the generated software especially in the automotive area has to fulfill very high safety requirements. In this paper, we present our project Methods of Model Quality1 in which we develop automated quality assurance methods for early development phases to improve the current unsatisfying situation. These methods comprise static analyses for domainspecific error detection, analyses to identify the most error-prone model parts through model metrics and furthermore slicing techniques for analyses support and result visualization. To estimate the model maintainability and changeability a quality model including architecture and design analyses is proposed as well. The expected results of our project will help to reduce development time and costs as well as to improve code quality and reliability. (12 refs)
Main heading: Static analysis
Controlled terms: Quality assurance
Uncontrolled terms: Code quality - Design Analysis - Development phasis - Development process - Development time - Domain specific - Embedded processors - Error prones - High safety - Model driven development - Model qualities - Project methods - Quality models
Classification Code: 723.1 Computer Programming - 913.3 Quality Assurance and Control
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

71. Software controlled memory layout reorganization for irregular array access patterns

Cho, Doosan (1); Issenin, Ilya (2); Dutt, Nikil (2); Yoon, Jonghee W. (1); Paek, Yunheung (1)
Source: *CASES'07: Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, p 179-188, 2007, *CASES'07: Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*; **ISBN-13:** 9781595938268; **DOI:** 10.1145/1289881.1289915; **Conference:** CASES'07: 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, September 30, 2007 - October 3, 2007; **Sponsor:** ACM Special Interest Group on Design Automation; ACM Special Interest Group on Embedded Systems; ACM SIG on Microarchitectural Research and Processing; **Publisher:** Association for Computing Machinery
Author affiliation: (1) School of EECS, Seoul National University (2) Center for Embedded Computer Systems, University of California, Irvine, CA 92697
Abstract: Many embedded array-intensive applications have irregular access patterns that are not amenable to static analysis for extraction of access patterns, and thus prevent efficient use of a Scratch Pad Memory (SPM) hierarchy for performance and power improvement. We present a profiling based strategy that generates a memory access trace which can be used to identify data elements with fine granularity that can profitably be placed in the SPMs to maximize performance and energy gains. We developed an entire toolchain that allows incorporation of the code required to profitably move data to SPMs; visualization of the extracted access pattern after profiling; and evaluation/ exploration of the generated application code to steer mapping of data to the SPM to yield performance and energy benefits.We present a heuristic approach that efficiently exploits the SPM using the profiler-driven access pattern behaviors. Experimental results on EEMBC and other industrial codes obtained with our framework show that we are able to achieve 36% energy reduction and reduce execution time by up to 22% compared to a cache based system. Copyright 2007 ACM. (30 refs)

Main heading: Computer software
Controlled terms: Access control - Conformal mapping - Control systems - Embedded systems - Energy utilization
Uncontrolled terms: Data layout - Scratch Pad Memory (SPM)
Classification Code: 525.3 Energy Utilization - 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 731.1 Control Systems - 921 Mathematics
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

72. Dynamic simulation of ray tracing for jitter and drift analysis

Wassom, Steven R. (1); Crowther, Blake G. (1)
Source: *Proceedings of SPIE - The International Society for Optical Engineering*, v 5420, p 97-105, 2004, *Modeling, Simulation, and Calibration of Space-based Systems*; **ISSN:** 0277786X; **DOI:** 10.1117/12.542175; **Conference:** Modeling, Simulation, and Calibration of Space-based Systems, April 15, 2004 - April 15, 2004; **Sponsor:** SPIE - The International Society for Optical Engineering; **Publisher:** SPIE
Author affiliation: (1) Space Dynamics Laboratory, Utah State University, 1695 N. Research Park Way, North Logan, UT 84341, United States
Abstract: Dynamic ray tracing is a new tool that combines optical ray tracing and dynamic simulation codes. The implementation presented in this paper is a customization of the commercial code ADAMS. The tool features a special subroutine that was written and linked to the code, enabling it to compute and display the paths and intersection points of reflected and refracted optical rays as the optical surfaces move dynamically. Its first intended use would be for analysis and control of high-frequency jitter and lower-frequency drift. In addition to "undesired" motions or deformations, the method may also be used to simulate intentionally moving optical components such as scanners or zoom systems. The main difference in this capability and that of the existing optical design codes is that this method yields visual dynamic results. In quasi-real time, the user can watch the ray trace move and the resultant image quality metric change due to unwanted or intentional motion of the optical elements. This approach will enable the user to more quickly understand and visualize the situation and will reduce the chances of error that arise when two codes have to be used (static ray tracing and dynamic simulation) to analyze the system. (3 refs)
Main heading: Ray tracing
Controlled terms: Algorithms - Computational geometry - Computer aided design - Computer simulation - Computer software - Dynamic mechanical analysis - Image analysis - Image quality - Jitter - Patents and inventions
Uncontrolled terms: Automated dynamic analysis of mechanical systems (ADAMS) - Drift - Dynamic automated ray tracing (DART) - Refracted optical rays
Classification Code: 716 Telecommunication; Radar, Radio and Television - 723.2 Data Processing and Image Processing - 723.5 Computer Applications - 741.1 Light/Optics - 901.3 Engineering Research - 921.4 Combinatorial Mathematics, Includes Graph Theory, Set Theory
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

73. 2nd International Conference on Software and Data Technologies/Evaluation of Novel Approaches to Software Engineering 2007, CSOFT/ENASE 2007

Source: *Communications in Computer and Information Science*, v 22 CCIS, 2008, *Software and Data Technologies - Second International Conference, ICSOFT/ENASE 2007, Revised Selected Papers*; **ISSN:** 18650929; **ISBN-10:** 3540886540, **ISBN-13:** 9783540886549; **Conference:** 2nd International Conference on Software and Data Technologies, ICSOFT 2007, July 22, 2007 - July 25, 2007; **Sponsor:** Workflow Management Coalition (WfMC); **Publisher:** Springer Verlag
Abstract: The proceedings contain 31 papers. The special focus in this conference is Software and Data Technologies. The topics include: benefits of enterprise ontology for the development of ICT-based value networks; SOA pragmatism; a simple language for novel visualizations of information; generic components for static operations at object level; a visual dataflow language for image segmentation and registration; a debugger for the interpreter design pattern; concepts for high-perfomance scientific computing; a model driven architecture approach to web development; reverse-architecting legacy software based on roles; a supporting tool for requirements elicitation using a domain ontology; pattern detection in object-oriented source code; testing the effectiveness of MBIUI life-cycle framework for the development of affective interfaces; an ontological SW architecture supporting agile development of semantic portals; the vcodex platform for data compression; classification of benchmarks for the evaluation of grid resource

planning algorithms; a disconnection-aware mechanism to provide anonymity in two-level P2P systems; approximation and scoring for XML data management; quantitative analysis of the top ten Wikipedia's; a semantic web approach for ontological instances analysis; aspects based modeling of web applications to support co-evolution; recommending trustworthy knowledge in KMS by using agents; recent developments in automated inferencing of emotional state from face images; inconsistency-tolerant integrity checking for knowledge assimilation; improving cutting-stock plans with multi-objective genetic algorithm; knowledge purpose and visualization; empirical experimentation for validating the usability of knowledge; an ontological investigation in the field of computer programs; formal problem domain modeling within MDA; model based testing for agent systems and a metamodel for defining development methodologies.
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

74. Performance tools for parallel programming

Mohr, Bernd (1); Wolf, Felix (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 4192 LNCS, p 8-9, 2006, *Recent Advances in Parallel Virtual Machine and Message Passing Interface - 13th European PVM/MPI User's Group Meeting, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 354039110X, **ISBN-13:** 9783540391104; **Conference:** 13th European PVM/MPI User's Group Meeting, September 17, 2006 - September 20, 2006; **Sponsor:** Etnus; IBM; Intel; NEC; Dolphin Interconnect Solutions; **Publisher:** Springer Verlag
Author affiliation: (1) Research Centre Jülich, Jülich, Germany
Abstract: Application developers are facing new and more complicated performance tuning and optimization problems as architectures become more complex. In order to achieve reasonable performance on these systems, HPC users need help from performance analysis tools. In this tutorial we will introduce the principles of experimental performance instrumentation, measurement, and analysis, with an overview of the major issues, techniques, and resources in performance tools development, as well as an overview of the performance measurement tools available from vendors and research groups. The focus of this tutorial will be on experimental performance analysis, which is currently the method of choice for tuning large-scale, parallel systems. The goal of experimental performance analysis is to provide the data and insights required to optimize the execution behavior of applications or system components. Using such data and insights, application and system developers can choose to optimize software and execution environments along many axes, including execution time, memory requirements, and resource utilization. In this tutorial we will present a broad range of techniques used for the development of software for performance measurement and analysis of scientific applications. These techniques range from mechanisms for simple code timings to multi-level hardware/software measurements. In addition, we will present state of the art tools from research groups, as well as software and hardware vendors, including practical tips and tricks on how to use them for performance tuning. When designing, developing, or using a performance tool, one has to decide on which instrumentation technique to use. We will cover the main instrumentation techniques, which can be divided into either static, during code development, compilation, or linking, or dynamic, during execution. The most common instrumentation approach augments source code with calls to specific instrumentation libraries. During execution, these library routines collect behavioral data. One example of static instrumentation systems that will be covered in details is the MPI profiling interface, which is part of the MPI specification, and was defined to provide a mechanism for quick development of performance analysis system for parallel programs. In addition, we will present similar work (POMP, OPARI) that has been proposed in the context of OpenMP. In contrast to static instrumentation, dynamic instrumentation allows users to interactively change instrumentation points, focusing measurements on code regions where performance problems have been detected. An example of such dynamic instrumentation systems is the DynInst project from the University of Maryland and University of Wisconsin, which provides an infrastructure to help tools developers to build performance tools. We will compare and contrast these instrumentation approaches. Regardless of the instrumentation mechanism, there are two dimensions that need to be considered for performance data collection: when the performance collection is triggered and how the performance data is recorded. The triggering mechanism can be activated by an external agent, such as a timer or a hardware counter overflow, or internally, by code inserted through instrumentation. The former is also known as sampling or asynchronous, while the latter is sometimes referred as synchronous. Performance data can be summarized during runtime and stored in the form of a profile, or can be stored in the form of traces. We will present these approaches and discuss how each one reflects a different balance among data volume, potential instrumentation perturbation, accuracy, and implementation complexity. Performance data should be stored in a format that allows the generality and extensibility necessary to represent a diverse set of performance metrics and measurement points, independent of language and architecture idiosyncrasies. We will describe common trace file formats (Vampir, CLOG, SLOG, EPILOG), as well as profile data formats based on the extensible Markup Language (XML), which is becoming a standard for describing performance data representation. Hardware performance counters have become an essential asset for application performance tuning. We will discuss in detail how users can access hardware performance

counters using application programming interfaces such as PAPI and PCL, in order to correlate the behavior of the application to one or more of the components of the hardware. Visualization systems should provide natural and intuitive user interfaces, as well as, methods for users to manipulate large data collections, such that they could grasp essential features of large performance data sets. In addition, given the diversity of performance data, and the fact that performance problems can arise at several levels, visualization systems should also be able to provide multiple levels of details, such that users could focus on interesting yet complex behavior while avoiding irrelevant or unnecessary details. We will discuss the different visualization and presentation approaches currently used on state of the art research tools, as well as tools from software and hardware vendors. The tutorial will be concluded with discussion on open research problems. Given the complexity of the state of the art of parallel applications, new performance tools must be deeply integrated, combining instrumentation, measurement, data analysis, and visualization. In addition, they should be able to guide or perform performance remediation. Ideally, these environments should scale to hundreds or thousands of processors, support analysis of distributed computations, and be portable across a wide range of parallel systems. Also, performing a whole series of experiments (studies) should be supported to allow a comparative or scalability analysis. We will discuss research efforts in automating the process of performance analysis such as the projects under the APART working group effort. We conclude the tutorial with a discussion on issues related to analysis of grid applications. © Springer-Verlag Berlin Heidelberg 2006.
Main heading: Parallel processing systems
Controlled terms: Computer aided design - Computer programming - Computer software - Data acquisition - Perturbation techniques - Systems analysis - Visualization - XML
Uncontrolled terms: Behavioral data - Experimental performance analysis - Implementation complexity - MPI profiling interface
Classification Code: 723.1 Computer Programming - 723.2 Data Processing and Image Processing - 723.5 Computer Applications - 912.3 Operations Research - 921 Mathematics
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

75. Identifying structural features of java programs by analysing the interaction of classes at runtime

Smith, Michael P. (1); Munro, Malcolm (1)
Source: *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, p 108-113, 2005, *Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*; **ISBN-10:** 0780395409, **ISBN-13:** 9780780395404; **DOI:** 10.1109/VISSOF.2005.1684316; **Article number:** 1684316; **Conference:** 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2005, September 25, 2005 - September 25, 2005; **Sponsor:** IEEE Computer Society; **Publisher:** IEEE Computer Society
Author affiliation: (1) Visualisation Research Group, Department of Computer Science, University of Durham, Durham, DHI 3LE, United Kingdom
Abstract: This paper describes research on visualising Java software at runtime in order to enable the identification of structural features. The aim is to highlight both the static and dynamic structure of the software and aid software engineers in tasks requiring program comprehension of the code. The paper takes the position that this type of analysis and visualisation for object oriented languages must be carried out with dynamic runtime information and that it cannot, in general, be obtained by static analysis alone. A case study is worked through to demonstrate the approach. © 2005 IEEE. (8 refs)
Main heading: Computer software
Controlled terms: Java programming language - Static analysis
Uncontrolled terms: Dynamic structure - Java program - Java software - Program comprehension - Run-time information - Runtimes - Structural feature
Classification Code: 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

76. Fault localization for null pointer exception based on stack trace and program slicing

Jiang, Shujuan (1); Li, Wei (1); Li, Haiyang (1); Zhang, Yanmei (1); Zhang, Hongchang (1); Liu, Yingqi (1)
Source: *Proceedings - International Conference on Quality Software*, p 9-12, 2012, *Proceedings - 12th International Conference on Quality Software, QSIC 2012*; **ISSN:** 15506002; **ISBN-13:** 9780769548333; **DOI:** 10.1109/

QSIC.2012.36; **Article number:** 6319219; **Conference:** 12th International Conference on Quality Software, QSIC 2012, August 27, 2012 - August 29, 2012; **Sponsor:** Northwest Polytechnic University; The University of Hong Kong; University of L'Aquila; **Publisher:** IEEE Computer Society
Author affiliation: (1) School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, China
Abstract: Null pointer exception is a commonly occurring error in Java programs, and many static analysis tools can identify such errors. However, most of existing tools are pure static analysis and suffer from the common problems of the pure static approaches. In this paper, we present a new approach for identifying null dereferences by combining the dynamically generated information (from the stack trace) with the static analysis. Starting at a dereference statement, where the null pointer exception occurred, our approach performs a backward program slicing guided by the stack trace. Then it performs the null identifying analysis and alias analysis on the sliced program. The approach also visualizes the analysis results and the related source codes. Finally, the paper also presents an implementation of the null pointer exception analysis. The results show the advantage of our approach for locating null pointer exception. © 2012 IEEE. (12 refs)
Main heading: Static analysis
Controlled terms: Computer software - Java programming language
Uncontrolled terms: Alias analysis - Exception analysis - Fault localization - Java program - Null pointer exception - Program slicing - Source codes - Stack trace - Static approach
Classification Code: 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

77. Design of a Parser for Real-Time Process Algebra

Zhao, Jianhua (1, 2); Wang, Yingxu (3)
Source: *Canadian Conference on Electrical and Computer Engineering*, v 2, p 1259-1262, 2003; **ISSN:** 08407789; **DOI:** 10.1109/CCECE.2003.1226128; **Conference:** CCECE 2003 Canadian Conference on Electrical and Computer Engineering: Toward a Caring and Humane Technology, May 4, 2003 - May 7, 2003; **Publisher:** Institute of Electrical and Electronics Engineers Inc.
Author affiliation: (1) Univ. of Calgary, Calgary, Alta., Canada (2) Stt. Key Lab. Novel Software T., Dept. of Computer Science, Nanjing University, Jiangsu 210093, China (3) Theor./Empirical Software Eng. R.C., Dept. of Electrical Engineering, University of Calgary, 2500 University Drive, NW, Calgary, Alta., T2N 1N4, Canada
Abstract: The Real-Time Process Algebra (RTPA) is a set of new mathematical notations for formally describing software system architectures, and static and dynamic behaviors. To bring RTPA into industrial software development practice, tools are needed for analyzing and visualizing RTPA specifications. The first step to develop the supporting tools is to build a grammar parser for recognizing the RTPA notation system. In this paper, a parser of RTPA is described. The parser takes a textual RTPA specification as input, and generates an Abstract Syntax Tree (AST) as its output. The generated AST represents RTPA tokens and lexical information in a structured format, which provides a foundation for further semantic analysis, code generation, visualization, and validation. (7 refs)
Main heading: Real time systems
Controlled terms: Computer programming - Computer software - Visualization
Uncontrolled terms: Parsers
Classification Code: 722.4 Digital Computers and Systems - 723.1 Computer Programming
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

78. Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis

Gu, Ruirui (1); Janneck, Jörn W. (2); Bhattacharyya, Shuvra S. (1); Raulet, Mickaël (3); Wipliez, Matthieu (3); Plishker, William (1)
Source: *IEEE Transactions on Circuits and Systems for Video Technology*, v 19, n 11, p 1646-1657, November 2009; **ISSN:** 10518215; **DOI:** 10.1109/TCSVT.2009.2031517; **Article number:** 5229343; **Publisher:** Institute of Electrical and Electronics Engineers Inc.
Author affiliation: (1) Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742, United States (2) Xilinx Inc., San Jose, CA 95124, United States (3) Institute of Electronics and Telecommunications, Rennes Laboratory, National Institute of Applied Sciences, Rennes Cedex 35043, France

Abstract: This paper presents an in-depth case study on dataflow-based analysis and exploitation of parallelism in the design and implementation of a MPEG reconfigurable video coding decoder. Dataflow descriptions have been used in a wide range of digital signal processing (DSP) applications, such as applications for multimedia processing and wireless communications. Because dataflow models are effective in exposing concurrency and other important forms of high level application structure, dataflow techniques are promising for implementing complex DSP applications on multicore systems, and other kinds of parallel processing platforms. In this paper, we use the client access license (CAL) language as a concrete framework for representing and demonstrating dataflow design techniques. Furthermore, we also describe our application of the differential item functioning dataflow interchange format package (TDP), a software tool for analyzing dataflow networks, to the systematic exploitation of concurrency in CAL networks that are targeted to multicore platforms. Using TDP, one is able to automatically process regions that are extracted from the original network, and exhibit properties similar to synchronous dataflow (SDF) models. This is important in our context because powerful techniques, based on static scheduling, are available for exploiting concurrency in SDF descriptions. Detection of SDF-like regions is an important step for applying static scheduling techniques within a dynamic dataflow framework. Furthermore, segmenting a system into SDF-like regions also allows us to explore cross-actor concurrency that results from dynamic dependences among different regions. Using SDF-like region detection as a preprocessing step to software synthesis generally provides an efficient way for mapping tasks to multicore systems, and improves the system performance of video processing applications on multicore platforms. © 2009 IEEE. (30 refs)
Main heading: Data flow analysis
Controlled terms: Computer software - Decoding - Digital signal processors - Interchanges - Motion Picture Experts Group standards - Multimedia signal processing - Multimedia systems - Parallel processing systems - Query languages - Response time (computer systems) - Scheduling - Signal processing - Visual communication - Wireless networks - Wireless telecommunication systems
Uncontrolled terms: CAL - Concurrency - Dataflow - Dataflow interchange format - MPEG RVC - Parallel processing
Classification Code: 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.2 Data Processing and Image Processing - 723 Computer Software, Data Handling and Applications - 723.3 Database Systems - 742.1 Photography - 912.2 Management - 723.5 Computer Applications - 722.4 Digital Computers and Systems - 716 Telecommunication; Radar, Radio and Television - 716.1 Information Theory and Signal Processing - 716.3 Radio Systems and Equipment - 406.1 Highway Systems - 716.4 Television Systems and Equipment - 717.1 Optical Communication Systems - 722.3 Data Communication, Equipment and Techniques - 717 Optical Communication
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

79. Proceedings - 13th European Conference on Software Maintenance and Reengineering, CSMR 2009

Source: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, 2009, *Proceedings - 13th European Conference on Software Maintenance and Reengineering, CSMR 2009*; **ISSN:** 15345351; **ISBN-13:** 9780769535890; **Conference:** 13th European Conference on Software Maintenance and Reengineering, CSMR 2009, March 24, 2009 - March 27, 2009; **Sponsor:** Reengineering Forum, REF; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Abstract: The proceedings contain 53 papers. The topics discussed include: design for maintenance - use of engineering principles and product line technology; Stevens lecture on software development methods at CSMR 2009; discovering comprehension pitfalls in class hierarchies; architectural complexity of large-scale software systems; software clustering using dynamic analysis and static dependencies; static security analysis based on input-related software faults; automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation; tool support for fault localization using architectural models; application of TreeNet in predicting object-oriented software maintainability: a comparative study; improving guidance when restructuring variabilities in software product lines; and behavioral pattern identification through visual language parsing and code instrumentation.
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

80. Computational tool for evaluation of seismic performance of reinforced concrete buildings

Kunnath, S.K. (1); Reinhorn, A.M. (1); Abel, J.F. (1)
Source: *Computers and Structures*, v 41, n 1, p 157-173, 1991; **ISSN:** 00457949; **DOI:** 10.1016/0045-7949(91)90165-I

Author affiliation: (1) State Univ of New York at Buffalo, Amherst, United States
Abstract: A special-purpose computational tool is developed to evaluate the inelastic seismic response of reinforced concrete buildings. A macromodel approach is used to analyze a discretized building composed of parallel frame-wall systems interconnected by transverse elements. The macro-behavioral models include the essential hysteretic characteristics of reinforced concrete sections and also account for the effects of distributed plasticity. All developments are incorporated into a computer code, IDARC. The program performs a series of tasks to enable a complete evaluation of the structural system: (a) monotonic analysis to establish the collapse mechanism and base shear capacity of the structure; (b) quasi-static cyclic analysis under force or deformation control; (c) transient dynamic analysis under horizontal and vertical seismic excitations; (d) reduction of the response quantities to damage indices so that a physical interpretation of the response is possible. The program is built around two graphical interfaces: one for preprocessing of structural and loading data; and the other for visualization of structural damage following the seismic analysis. The program can serve as an invaluable tool in estimating the seismic performance of existing reinforced concrete buildings and for designing new structures within acceptable levels of damage. (14 refs)
Main heading: Buildings
Controlled terms: Computer Graphics - Computer Software - Concrete Construction--Earthquake Resistance - Earthquake Resistance--Structural Analysis - Structural Analysis--Computer Aided Analysis
Uncontrolled terms: Software Package IDARC
Classification Code: 402 Buildings and Towers - 405 Construction Equipment and Methods; Surveying - 408 Structural Design - 412 Concrete - 484 Seismology - 723 Computer Software, Data Handling and Applications
Treatment: Applications (APP) - Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

81. ASE'10 - Proceedings of the IEEE/ACM International Conference on Automated Software Engineering

Source: *ASE'10 - Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2010, *ASE'10 - Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*; **ISBN-13:** 9781450301169; **Conference:** 25th IEEE/ACM International Conference on Automated Software Engineering, ASE'10, September 20, 2010 - September 24, 2010; **Sponsor:** SIGSOFT; SIGART; IEEE CS; **Publisher:** Association for Computing Machinery
Abstract: The proceedings contain 84 papers. The topics discussed include: analyzing security architectures; VikiBuilder: end-user specification and generation of visual Wikis; software design sketching with Calico; automatically documenting program changes; towards automatically generating summary comments for Java methods; automatic detection of nocuous coordination ambiguities in natural language requirements; flexible and scalable consistency checking on product line variability models; variability modeling in the real: a perspective from the operating systems domain; RESISTting reliability degradation through proactive reconfiguration; automatic construction of an effective training set for prioritizing static analysis warnings; an automated approach for finding variable-constant pairing bugs; deviance from perfection is a better criterion than closeness to evil when identifying risky code; and seamlessly integrated, but loosely coupled - building user interfaces from heterogeneous components.
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

82. Contract-based reasoning for verification and certification of secure information flow policies in industrial workflows

Hatcliff, John (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 5256 LNCS, p 3-4, 2008, *Formal Methods and Software Engineering - 10th International Conference on Formal Engineering Methods, ICFEM 2008, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 354088193X, **ISBN-13:** 9783540881933; **DOI:** 10.1007/978-3-540-88194-0-3; **Conference:** 10th International Conference on Formal Engineering Methods, ICFEM 2008, October 27, 2008 - October 31, 2008; **Sponsor:** ICFEM Organizing Committee; **Publisher:** Springer Verlag
Author affiliation: (1) SAnToS Laboratory, Kansas State University, Manhattan, KS 66506, United States
Abstract: Successful transfer of formal engineering methods from academia to industrial development depends on a variety of factors: a proper understanding of the industrial development context, effective and usable technology that can be integrated with development workflows to provide a compelling solution to serious development challenges, "buy-in" from industrial developers and management, an appropriate business model for supporting the deployed

technology, plus a lot of luck. I describe how many of these factors are manifesting themselves in an effort by our research group to transition rigorous static analyses and novel Hoare-style logics into a large industrial development process for information assurance and security applications. The applications that we are targeting address the following problem: international infrastructure and defense forces are increasingly relying on complex systems that share information with multiple levels of security (MLS). In such systems, there is a strong tension between providing aggressive information flow to gain operational and strategic advantage while preventing leakage to unauthorized parties. In this context, it is exceedingly difficult to specify and certify security policies, and produce evidence that a system provides end-to-end trust. In the past, verification and certification obligations in this domain have been met by using heavy-weight theorem proving technology that requires many manual steps or by light-weight contract-based static analyses that are too imprecise for specifying and verifying crucial information flow properties. In this talk, I will explain how our research team is (a) building integrated tool support for automatically discovering and visualizing information flows through programs and architectures, and (b) providing code-integrated software contracts for specifying information flow policies, and (c) applying synergistic blends of static analyses and automated reasoning based on weakest-precondition calculi to aid developers in automatically discharging verification obligations. These techniques aim to hit a "sweet spot" that provides greater automation and developer integration than previous theorem-proving-based approaches while offering increased precision over previous static-analysis-based frameworks. Throughout the presentation, I will assess approaches/strategies that have been successful in moving our research results into industrial practice and summarize challenges that remain. © 2008 Springer Berlin Heidelberg. (11 refs)
Main heading: Formal methods
Controlled terms: Automata theory - Biomineralization - Contracts - Problem solving - Security of data - Software engineering - Static analysis - Stream flow - Theorem proving - Verification
Uncontrolled terms: Automated reasonings - Business models - Complex systems - Defense forces - Do-mains - Engineering methods - Following problems - Industrial developments - Industrial practices - Information Assurance and securities - Information flow policies - Information flows - Infra structures - Integrated softwares - Integrated tools - Multiple levels - Research groups - Research results - Research teams - Secure information flows - Security policies - Strategic advantages - Sweet spots - Synergistic blends - Visualizing informations - Workflows
Classification Code: 723.2 Data Processing and Image Processing - 723.4 Artificial Intelligence - 723.5 Computer Applications - 801.2 Biochemistry - 902.3 Legal Aspects - 912.2 Management - 921 Mathematics - 723.1 Computer Programming - 407.2 Waterways - 461.8 Biotechnology - 461.9 Biology - 631.1 Fluid Flow, General - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 721.2 Logic Elements - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

83. Proceedings 11th Working Conference on Reverse Engineering, WCRE 2004

Source: *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2004, *Proceedings - 11th Working Conference on Reverse Engineering, WCRE 2004*; **ISSN:** 10951350; **Conference:** Proceedings - 11th Working Conference on Reverse Engineering, WCRE 2004, November 8, 2004 - November 12, 2004; **Sponsor:** IEEE Computer Society, Technical Council on Software Engineering; Reengineering Forum; **Publisher:** IEEE Computer Society
Abstract: The proceedings contain 38 papers from the 11th Working Conference on Reverse Engineering, WCRE 2004. The topics discussed include: model engineering for software modernization; static and dynamic analyses of programs with implicit control law; experiences with an industrial long-term reengineering project; using a decomplier for real-world source recovery; a novel software visualization model to support software comprehension; exploring software evolution using spectrographs; aspect mining through the formal concept analysis of execution traces; the small world of software reverse engineering; the efficiency of specification fragments; and towards an effective approach for reverse engineering.
Main heading: Reverse engineering
Controlled terms: COBOL (programming language) - Codes (symbols) - Computer operating systems - Computer software maintenance - Computer viruses - Database systems - File organization - Java programming language - Legacy systems - Medical computing - Network protocols - Object oriented programming - Program processors - Software engineering - Wireless telecommunication systems - XML
Uncontrolled terms: Coding patterns - Data reengineering - Decompilation - Dynamic analysis - EiReV - Model driven engineering (MDE) - Reverse engineering tools - Software modernization - Unified modeling language (UML) - Windows operating systems
Classification Code: 723.3 Database Systems - 723.2 Data Processing and Image Processing - 723.1.1 Computer Programming Languages - 723.5 Computer Applications - 723.1 Computer Programming - 716 Telecommunication;

Radar, Radio and Television - 461.1 Biomedical Engineering - 723 Computer Software, Data Handling and Applications
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

84. Second generation object-oriented development

De Carvalho, Sergio E.R. (1); Cruz, Sylvia De O. (1); De Oliveirae, Toacy C. (1)
Source: *Electronic Notes in Theoretical Computer Science*, v 14, p 94-106, 1998, *US-Brazil Joint Workshops on the Formal Foundations of Software Systems*; **ISSN:** 15710661; **DOI:** 10.1016/S1571-0661(05)80232-4; **Conference:** US-Brazil Joint Workshops on the Formal Foundations of Software Systems, May 11, 1997 - May 11, 1997; **Publisher:** Elsevier
Author affiliation: (1) Lab. de Metodos Formais Departamento de Informdtica Pont. Universidade Catolica Do Rio de Janeiro, Rua Marques de Sando Vicente 225, Rio de Janeiro, RJ, 22453-900, Brazil
Abstract: Well-integrated development tools, allowing automatic code generation from visual representations of analysis and design decisions, are important assets in handling the complexities of todays software. This paper describes a few features of a method independent, object-oriented development tool, essentially consisting of a users plane, where visual system construction takes place, a formal plane, where user actions are verified, and underlying platforms, providing user-formal mappings. The paper concentrates on the users plane, addressing aspects of static and dynamic system views. © 1998 Published by Elsevier Science B.V. (12 refs)
Main heading: Computer aided software engineering
Controlled terms: Codes (symbols) - Computer aided design - Error analysis - Mathematical models - Object oriented programming - Real time systems
Uncontrolled terms: Code generation - Object oriented systems - Quality codes - Visual systems
Classification Code: 722.4 Digital Computers and Systems - 723.1 Computer Programming - 723.2 Data Processing and Image Processing - 723.5 Computer Applications - 921.6 Numerical Methods
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

85. The MOLDY short-range molecular dynamics package

Ackland, G.J. (1); D'Mellow, K. (1); Daraszewicz, S.L. (1); Hepburn, D.J. (1); Uhrin, M. (1); Stratford, K. (1)
Source: *Computer Physics Communications*, v 182, n 12, p 2587-2604, December 2011; **ISSN:** 00104655; **DOI:** 10.1016/j.cpc.2011.07.014; **Publisher:** Elsevier
Author affiliation: (1) School of Physics and Astronomy, University of Edinburgh, King's Buildings, Edinburgh, EH9 3JZ, United Kingdom
Abstract: We describe a parallelised version of the MOLDY molecular dynamics program. This Fortran code is aimed at systems which may be described by short-range potentials and specifically those which may be addressed with the embedded atom method. This includes a wide range of transition metals and alloys. MOLDY provides a range of options in terms of the molecular dynamics ensemble used and the boundary conditions which may be applied. A number of standard potentials are provided, and the modular structure of the code allows new potentials to be added easily. The code is parallelised using OpenMP and can therefore be run on shared memory systems, including modern multicore processors. Particular attention is paid to the updates required in the main force loop, where synchronisation is often required in OpenMP implementations of molecular dynamics. We examine the performance of the parallel code in detail and give some examples of applications to realistic problems, including the dynamic compression of copper and carbon migration in an iron-carbon alloy. Program summary: Program title: MOLDY Catalogue identifier: AEJU-v1-0 Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEJU-v1-0.html Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland Licensing provisions: GNU General Public License version 2 No. of lines in distributed program, including test data, etc.: 382 881 No. of bytes in distributed program, including test data, etc.: 6 705 242 Distribution format: tar.gz Programming language: Fortran 95/OpenMP Computer: Any Operating system: Any Has the code been vectorised or parallelized?: Yes. OpenMP is required for parallel execution RAM: 100 MB or more Classification: 7.7 Nature of problem: Moldy addresses the problem of many atoms (of order 106) interacting via a classical interatomic potential on a timescale of microseconds. It is designed for problems where statistics must be gathered over a number of equivalent runs, such as measuring thermodynamic properities, diffusion, radiation damage, fracture, twinning deformation, nucleation and growth of phase transitions, sputtering etc. In the vast majority of materials, the interactions are non-pairwise, and the code must be able to deal with many-

body forces. Solution method: Molecular dynamics involves integrating Newton's equations of motion. MOLDY uses verlet (for good energy conservation) or predictor-corrector (for accurate trajectories) algorithms. It is parallelised using open MP. It also includes a static minimisation routine to find the lowest energy structure. Boundary conditions for surfaces, clusters, grain boundaries, thermostat (Nose), barostat (Parrinello-Rahman), and externally applied strain are provided. The initial configuration can be either a repeated unit cell or have all atoms given explictly. Initial velocities are generated internally, but it is also possible to specify the velocity of a particular atom. A wide range of interatomic force models are implemented, including embedded atom, Morse or Lennard-Jones. Thus the program is especially well suited to calculations of metals. Restrictions: The code is designed for short-ranged potentials, and there is no Ewald sum. Thus for long range interactions where all particles interact with all others, the order-N scaling will fail. Different interatomic potential forms require recompilation of the code. Additional comments: There is a set of associated open-source analysis software for postprocessing and visualisation. This includes local crystal structure recognition and identification of topological defects. Running time: A set of test modules for running time are provided. The code scales as order N. The parallelisation shows near-linear scaling with number of processors in a shared memory environment. A typical run of a few tens of nanometers for a few nanoseconds will run on a timescale of days on a multiprocessor desktop. © 2011 Elsevier B.V. (44 refs)
Main heading: Molecular dynamics
Controlled terms: Application programming interfaces (API) - Atoms - Boundary conditions - Computer operating systems - Computer systems programming - Crystal structure - Distributed computer systems - Dynamics - Embedded systems - Equations of motion - FORTRAN (programming language) - Grain boundaries - Growth (materials) - Metallurgy - Multicore programming - Parallel processing systems - Phase transitions - Problem oriented languages - Radiation damage - Random access storage - Software testing - Transition metals - Visualization
Uncontrolled terms: Applied strain - Carbon migration - Catalogue identifiers - Distributed program - Dynamic compression - Embedded atoms - Embedded-atom method - FORTRAN codes - GNU general public license - Initial configuration - Initial velocities - Interatomic forces - Interatomic potential - Ireland - Iron-carbon alloys - Lennard jones - Long range interactions - Lowest energy structure - Many-body - Modular structures - Molecular-dynamics ensemble - Multi-core processor - Nucleation and growth - Open-source - OpenMP - Parallel code - Parallel executions - Parallelisation - Parrinello-Rahman - Predictor corrector - Programming language - Recompilation - Repeated unit cells - Running time - Shared memories - Shared memory system - Solution methods - Standard potential - Structure recognition - Test data - Test modules - Time-scales - Topological defect - Transition metals and alloys
Classification Code: 951 Materials Science - 931.3 Atomic and Molecular Physics - 931.1 Mechanics - 921 Mathematics - 902.1 Engineering Graphics - 801.4 Physical Chemistry - 723 Computer Software, Data Handling and Applications - 722.4 Digital Computers and Systems - 722.1 Data Storage, Equipment and Techniques - 622.2 Radiation Effects - 531 Metallurgy and Metallography
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

86. 2009 31st International Conference on Software Engineering - Proceedings, ICSE 2009

Source: *Proceedings - International Conference on Software Engineering*, 2009, *2009 31st International Conference on Software Engineering - Proceedings, ICSE 2009*; **ISSN:** 02705257; **ISBN-13:** 9781424434527; **Conference:** 2009 31st International Conference on Software Engineering, ICSE 2009, May 16, 2009 - May 24, 2009; **Sponsor:** ACM - Association for Computing Machinery; IEEE; IEEE Computer Society; SIGSOFT - Special Interest Group on Software Engineering; **Publisher:** IEEE Computer Society
Abstract: The proceedings contain 70 papers. The topics discussed include: predicting build failures using social network analysis on developer communication; how tagging helps bridge the gap between social and technical aspects in software development; tesseract: interactive visual exploration of socio-technical relationships in software development; taming coincidental correctness: coverage refinement with context patterns to improve fault localization; lightweight fault-localization using multiple coverage types; succession: measuring transfer of code and developer productivity; predicting faults using the complexity of code changes; a case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment; model evolution by run-time parameter adaptation; the road not taken: estimating path execution frequency statically; and automatic dimension inference and checking for object-oriented programs.
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

87. Supporting program development comprehension by visualising iterative design

Boisvert, Charles (1)

Source: *Proceedings of the International Conference on Information Visualization*, v 8, p 717-722, 2004, *Proceedings - Eighth International Conference on Information Visualisation, IV 2004*; **ISSN:** 10939547; **Conference:** Proceedings - Eighth International Conference on Information Visualisation, IV 2004, July 14, 2004 - July 16, 2004; **Sponsor:** IEEE Computer Society; **Publisher:** Institute of Electrical and Electronics Engineers Inc.
Author affiliation: (1) City College Norwich
Abstract: eL-CID (e-Learning by Communicating Iterative Design) demonstrates computer programs' iterative design using computer animation. It translates descriptions of iterative editing into an animated demonstration. An analysis of the work of expert programming trainers shows that successive versions of a program are shown statically. eL-CID attempts to visualise the changes dynamically as if code was being edited in front of the user. Several example demonstrations have been developed. To the author's knowledge, this is the first system designed to visualise the iterative process of program development. (23 refs)
Main heading: Learning systems
Controlled terms: Algorithms - Animation - Computer programming - Human computer interaction - Iterative methods - Software engineering
Uncontrolled terms: E-learning - Iterative development - Software visualization
Classification Code: 723.1 Computer Programming - 723.5 Computer Applications - 921.6 Numerical Methods
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

88. ILP Platform Optimization of a YAPI Parallel H.264/AVC encoder

Ammari, Ahmed Chiheb (1); Jemai, Abderrazek (1); Zrida, Hajer Krichene (2); Abid, Mohamed (2)
Source: *2008 2nd International Conference on Signals, Circuits and Systems, SCS 2008*, 2008, *2008 2nd International Conference on Signals, Circuits and Systems, SCS 2008*; **ISBN-13:** 9781424426287; **DOI:** 10.1109/ICSCS.2008.4746911; **Article number:** 4746911; **Conference:** 2008 2nd International Conference on Signals, Circuits and Systems, SCS 2008, November 7, 2008 - November 9, 2008; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) National Institute of Applied Sciences and Technology, (INSAT), 7 November- Carthage University, Tunisia (2) Electrical Engineering Department, CES Laboratory, ENIS Institute, Tunisia
Abstract: The H.264/AVC (Advanced Video Codec) new video coding standard provides higher coding efficiency relative to former standards at the expense of higher computational requirements. Given the potential applications of this technology, we are developing an application environment able to decode an MPEG2 stream, convert it into an H.264 stream, and stream it over a network. This paper focuses on the H.264 video encoder implementation. The absolute complexity of the obtained costefficient configuration outlined the potential of using a multiple processors platform for executing a parallel code version of the H.264 reference software. For this, a starting YAPI parallel Kahn Process Network (KPN) model is proposed. This model has been implemented and validated at a high system-level using the YAPI multi-threading programming interface. To identify the potential bottlenecks of the starting parallel model, communication and computation workload analysis are considered. Based on this analysis, an optimized parallel YAPI/KPN model with maximum workload balance is provided. For cost-effective realization, mapping the validated parallel model on the STMicroelectronics mb392 multiprocessor platform is motivated. For this purpose, a static code parser for the ST220 Very Large Instruction Word (VLIW) processor is developed to analyze, for each process of the model, the instruction level parallelism (ILP) effectively used by the ST220 cross compiler. Using this tool, the binary code of each process, cross-compiled for an ST220, is statically analyzed and the processes demanding further low level optimization are identified. To maximize the ILP for the ST220 VLIW architecture, a low-level algorithmic optimization of the motion estimation and compensation process is performed. ©2008 IEEE. (17 refs)
Main heading: Motion estimation
Controlled terms: Binary codes - Image coding - Motion Picture Experts Group standards - Optimization - Program compilers - Very long instruction word architecture - Visual communication
Uncontrolled terms: Advanced video codecs - Algorithmic optimizations - Application environments - Coding efficiencies - Computational requirements - Cost-efficient - Estimation and compensations - H.264 video encoders - H.264/AVC - Instruction level parallelisms - Kahn process networks - Low levels - Multi-processor platforms - Multi-threading - Multiple processors - Parallel codes - Parallel models - Potential applications - Programming interfaces - Reference softwares - St microelectronics - Static codes - System levels - Very large instruction words - Video coding standards - Vliw architectures - Workload analysis - Workload balances
Classification Code: 742.1 Photography - 741 Light, Optics and Optical Devices - 723.2 Data Processing and Image Processing - 723.1 Computer Programming - 921.5 Optimization Techniques - 723 Computer Software, Data Handling and Applications - 717.1 Optical Communication Systems - 716.4 Television Systems and Equipment - 716.1 Information Theory and Signal Processing - 722 Computer Systems and Equipment

Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

89. Using interactive visualizations of WWW log data to characterize access patterns and inform site design

Hochheiser, Harry (1); Shneiderman, Ben (2)
Source: *Journal of the American Society for Information Science and Technology*, v 52, n 4, p 331-343, Febrary 15, 2001; **ISSN:** 15322882; **DOI:** 10.1002/1532-2890(2000)9999:9999<::AID-ASI1066>3.0.CO;2-Y; **Publisher:** John Wiley and Sons Inc.
Author affiliation: (1) Human-Computer Interaction Lab, Department of Computer Science, University of Maryland, College Park, MD 20742, United States (2) Department of Computer Science, Institute for Systems Research and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, United States
Abstract: HTTP server log files provide Web site operators with substantial detail regarding the visitors to their sites. Interest in interpreting this data has spawned an active market for software packages that summarize and analyze this data, providing histograms, pie graphs, and other charts summarizing usage patterns. Although useful, these summaries obscure useful information and restrict users to passive interpretation of static displays. Interactive visualizations can be used to provide users with greater abilities to interpret and explore Web log data. By combining two-dimensional displays of thousands of individual access requests, color, and size coding for additional attributes, and facilities for zooming and filtering, these visualizations provide capabilities for examining data that exceed those of traditional Web log analysis tools. We introduce a series of interactive visualizations that can be used to explore server data across various dimensions. Possible uses of these visualizations are discussed, and difficulties of data collection, presentation, and interpretation are explored. (26 refs)
Main heading: Information technology
Controlled terms: Data processing - Interactive computer systems - Statistical methods - Two dimensional - User interfaces - Websites - World Wide Web
Uncontrolled terms: Access pattern - Inform site design - Interactive visualization - Log data
Classification Code: 722.2 Computer Peripheral Equipment - 722.4 Digital Computers and Systems - 723.2 Data Processing and Image Processing - 723.5 Computer Applications - 922.2 Mathematical Statistics
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

90. Resource usage contracts for .NET

Tapicer, Jonathan (1); Garbervetsky, Diego (1); Rouaux, Martin (1)
Source: *Proceedings - International Conference on Software Engineering*, p 56, 2011, *TOPI'11 - Proceedings of the 1st Workshop on Developing Tools as Plug-Ins, Co-located with ICSE 2011*; **ISSN:** 02705257; **ISBN-13:** 9781450305990; **DOI:** 10.1145/1984708.1984725; **Conference:** 1st Workshop on Developing Tools as Plug-ins, TOPI 2011, Co-located with ICSE 2011, May 28, 2011 - May 28, 2011; **Sponsor:** ACM SIGSOFT; IEEE CS; **Publisher:** IEEE Computer Society
Author affiliation: (1) Departamento de Computación, FCEyN, UBA, Buenos Aires, Argentina
Abstract: CODE CONTRACTS [2] is a tool that allows the specification and verification of contracts (pre, post-condition, invariants) in all .NET based programming languages. RESOURCE CONTRACTS is an extension of this language to specify resource usage in .NET programs. The new annotations, initially focussed on dynamic memory, enable modular analysis of both memory consumption and lifetime properties. They are checked by relying on the own CODE CONTRACTS static verifier and a points-to analysis. This approach is implemented as a VISUAL STUDIO extension1, providing facilities such us autocompletion and verification at build time. Copyright 2011 ACM. (3 refs)
Uncontrolled terms: Build time - Memory consumption - Modular analysis - ON dynamics - Points-to analysis - Resource usage - Specification and verification - Static verification - Visual studios
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

91. Using Verilog LOGISCOPE to analyze student programs

Mengel, Susan A. (1); Ulans, Joseph (1)

Source: *Proceedings - Frontiers in Education Conference*, v 3, p 1213-1218, 1998; **ISSN:** 01905848; **Conference:** Proceedings of the 1998 28th Annual Frontiers in Education Conference, FIE. Part 3 (of 3), November 4, 1998 - November 7, 1998; **Sponsor:** IEEE; **Publisher:** IEEE
Author affiliation: (1) Texas Tech Univ, Lubbock, United States
Abstract: It is difficult for teachers and graders to give an in-depth evaluation of student programs to the point of checking every line of code due to the amount of time checking would take. The difficulty worsens when a typical introductory programming course may have over 100 students. Solutions to this difficulty may involve only checking to see if the program executes correctly (dynamic analysis), glancing over the program to see if appropriate documentation is present (static analysis), and glancing over the code for any problems (static analysis). Automated solutions are difficult to construct as can be seen by the fact that a few exist in homegrown versions (which typically are not available for general use and few examples are given in the literature) and commercial solutions are in the thousands of dollars. One commercial solution, however, is Verilog LOGISCOPE which offers a limited number of licenses free to educators. LOGISCOPE is a static analysis checker capable of taking hundreds of individual measurements of a program, such as lines of code, McCabe's cyclomatic complexity, and number of operators. It also shows the control flow graph of a program which is a depiction of the statements, if structures, and looping structures in a program. LOGISCOPE enables the complexity and quality of a program to be analyzed yielding valuable feedback to both students and educators. It allows visualization of the measurements taken through the control flow graphs and Kiveat diagrams. The operation of LOGISCOPE is shown by using typical student programs taken from the introductory computing course at Texas Tech University. Then the results of analyzing several programs from the same class are given to show the diversity of results. Finally, how LOGISCOPE can be used in education to help students improve their programming and help instructors evaluate programs better is considered. (8 refs)
Main heading: Engineering education
Controlled terms: Program debugging - Teaching
Uncontrolled terms: Software package Verilog LOGISCOPE
Classification Code: 723.1 Computer Programming - 901.2 Education
Treatment: Applications (APP) - General review (GEN)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

92. Improved Adaptive Interpolation Filter for H.264/AVC

Wei, Yuping (1); Ji, Xiangyang (1); Zhang, Naiyao (1); Dai, Qionghai (1)
Source: *Tsinghua Science and Technology*, v 15, n 2, p 216-220, April 2010; **ISSN:** 10070214; **DOI:** 10.1016/S1007-0214(10)70053-5; **Publisher:** Tsinghua University Press
Author affiliation: (1) Tsinghua National Laboratory for Information Science and Technology, Department of Automation, Tsinghua University, Beijing, 100084, China
Abstract: Motion compensation with adaptive interpolation filters (AIF) was developed to compensate for temporary varieties in the aliasing of video signals and improve the coding efficiency. The AIF achieves better RD performance than common static interpolation filtering by exploiting the statistics in the reference frames' local auto-correlation and the local cross-correlation between the current encoding frame and the reference frames. This paper presents an interpolation filter buffering structure that derives the current encoding frame's interpolation filters from the filters of previous frames. The number of encoding bits for the filter coefficients is reduced by encoding only the differences between the current filter coefficients and the corresponding buffered filter coefficients. Experimental results show that in comparison with the AIF in the current "key technical area" (KTA) reference software, this interpolation filter buffering structure further improves all the test sequences (with up to 2.87% bit rate saving) with negligible computational increase. © 2010 Tsinghua University Press. (13 refs)
Main heading: Interpolation
Controlled terms: Cluster analysis - Discriminant analysis - Encoding (symbols) - Image coding - Motion compensation - Motion Picture Experts Group standards - Regression analysis - Signal filtering and prediction - Visual communication
Uncontrolled terms: Adaptive interpolation - adaptive interpolation filter - Aliasing - Bit-rate savings - Coding efficiency - Cross correlations - Filter coefficients - H.264/AVC - Interpolation filtering - Interpolation filters - Local auto-correlation - Reference frame - Reference software - Test sequence - Video coding - Video signal
Classification Code: 922 Statistical Methods - 921.6 Numerical Methods - 903.1 Information Sources and Analysis - 742.1 Photography - 741 Light, Optics and Optical Devices - 922.2 Mathematical Statistics - 731.1 Control Systems - 723 Computer Software, Data Handling and Applications - 717.1 Optical Communication Systems - 716.4 Television Systems and Equipment - 716.1 Information Theory and Signal Processing - 723.2 Data Processing and Image Processing
Database: Compendex

Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

93. Titus on embedded: Real-time java hits its stride

Titus, Jon
Source: *ECN Electronic Component News*, v 53, n 14, p 12-13, December 2009; **ISSN:** 15233081; **Publisher:** Omeda
Abstract: Java, an object-oriented language, is encouraging developers to build programs with strong encapsulation isolation functions from other part of a system that makes it easy to create software from independent components. The developers need high-level tools such as static analysis tool from Coverity, a source-code-analyzer from SofCheck, or Klocwork tools that let developers to visualize interactions and relationship between parts of a large software system. An off-the shelf Java virtual machine (VM) is not appropriate for real-time embedded programming as it does not specifically honor the priority of threads that a real time Java VM must honor. The Aonix Perc Ultra Java VM performs garbage collection in real time ensuring only short periods away from the application. Dr. Kelvin Nilsen, chief technology officer at Aonix, explains that Java works well when the complex and large software projects are managed properly.
Main heading: Java programming language
Controlled terms: Complexation - Computer software - Object oriented programming - Refuse collection - Waste disposal
Uncontrolled terms: Chief technology officers - Embedded programming - Garbage collection - Independent components - Java virtual machines - Large software systems - Object-oriented languages - Real time - Real-time Java - Short periods - Software project
Classification Code: 452 Municipal and Industrial Wastes; Waste Treatment and Disposal - 452.4 Industrial Wastes Treatment and Disposal - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 802.2 Chemical Reactions
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

94. Representing and integrating dynamic collaborations in IDEs

Röthlisberger, David (1); Greevy, Orla (1)
Source: *Proceedings - Working Conference on Reverse Engineering, WCRE*, p 74-78, 2008, *Proceedings - 15th Working Conference on Reverse Engineering, WCRE 2008*; **ISSN:** 10951350; **ISBN-10:** 0769534295, **ISBN-13:** 9780769534299; **DOI:** 10.1109/WCRE.2008.53; **Article number:** 4656396; **Conference:** 15th Working Conference on Reverse Engineering, WCRE 2008, October 15, 2008 - October 18, 2008; **Sponsor:** Reengineering Forum; Fonds de la Recherche Scientifique; Universiteit Antwerpen; Universita degli Studi del Sannio; **Publisher:** IEEE Computer Society
Author affiliation: (1) Software Composition Group, University of Bern, Switzerland
Abstract: Static views of object-oriented source code as presented in a development environment (IDE) do not provide explicit representations of dynamic collaboration to describe how source artifacts communicate at runtime. Direct access within an IDE to explicit representations of dynamic collaborations would provide developers with useful insights into a system's behavior. In this paper we describe how we seamlessly integrate novel interactive visual representations of dynamic collaborations between static artifacts to complement traditional static concepts within the IDE. We motivate our work and introduce our enhancements in our prototype IDE (Hermion) and provide validation for our work by means of case studies and benchmarks. © 2008 IEEE. (12 refs)
Main heading: Dynamic analysis
Controlled terms: Integrodifferential equations - Reengineering - Reflection - Reverse engineering
Uncontrolled terms: Case studies - Development environments - Direct accesses - Dynamic collaborations - Explicit representations - Object-oriented - Partial behavioral reflection - Program comprehension - Source codes - Visual representations
Classification Code: 921.2 Calculus - 913.3 Quality Assurance and Control - 912.2 Management - 741.1 Light/Optics - 723 Computer Software, Data Handling and Applications - 711 Electromagnetic Waves - 422.2 Strength of Building Materials : Test Methods
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

95. Chava: Reverse engineering and tracking of Java applets

Korn, Jeffrey (1); Chen, Yih-Farn (1); Koutsofios, Eleftherios (1)

Source: *Reverse Engineering - Working Conference Proceedings*, p 314-325, 1999; **Conference:** Proceedings of the 1999 6th Working Conference on Reverse Engineering (WCRE'99), October 6, 1999 - October 8, 1999; **Sponsor:** IEEE Computer Society; **Publisher:** IEEE
Author affiliation: (1) Princeton Univ, Princeton, United States
Abstract: Java applets have been used increasingly on web sites to perform client-side processing and provide dynamic content. While many web site analysis tools are available, their focus has been on static HTML content and most ignore applet code completely. This paper presents Chava, a system that analyzes and tracks changes in Java applets. The tool extracts information from applet code about classes, methods, fields and their relationships into a relational database. Supplementary checksum information in the database is used to detect changes in two versions of a Java applet. Given our Java data model, a suite of programs that query, visualize, and analyze the structural information were generated automatically from CIAO, a retargetable reverse engineering system. Chava is able to process either Java source files or compiled class files, making it possible to analyze remote applets whose source code is unavailable. The information can be combined with HTML analysis tools to track both the static and dynamic content of many web sites. This paper presents our data model for Java and describes the implementation of Chava. Advanced reverse engineering tasks such as reachability analysis, clustering, and program differencing can be built on top of Chava to support design recovery and selective regression testing. In particular, we show how Chava is used to compare several Java Development Kit (JDK) versions to help spot changes that might impact Java developers. Performance numbers indicate that the tool scales well. (30 refs)
Main heading: Reverse engineering
Controlled terms: Computer aided software engineering - Computer architecture - Data structures - HTML - Java programming language - Mathematical models - Relational database systems - World Wide Web
Uncontrolled terms: Retargetable reverse engineering systems - Software package Chava - Supplementary checksum informations
Classification Code: 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.3 Database Systems - 723.5 Computer Applications
Treatment: Applications (APP) - Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

96. Proceedings of the 1997 Canadian Conference on Electrical and Computer Engineering, CCECE'97. Part 1 (of 2)

Source: *Canadian Conference on Electrical and Computer Engineering*, v 1, 1997; **ISSN:** 08407789; **Conference:** Proceedings of the 1997 Canadian Conference on Electrical and Computer Engineering. CCECE'97. Part 2 (of 2), May 25, 1997 - May 28, 1997; **Sponsor:** IEEE; **Publisher:** IEEE
Abstract: The proceedings contains 95 papers from 1997 IEEE Canadian Conference on Electrical and Computer Engineering. Topics discussed include: multimedia communication; state machines; circuit partitioning; computer-aided designs; automatic generation control; autonomous wind-diesel system; smart compensators; automatic voltage regulators; pattern recognition systems; neural networks; error correcting codes; digital filters; delta-sigma modulators; video communication; image communication; digital arithmetic; three-phase power transformer; virtual reality; robots; digital signal processing; software engineering; cryptography; and applied electromagnetics.
Main heading: Computer applications
Controlled terms: Coding errors - Computer aided network analysis - Computer simulation - Digital communication systems - Electric power systems - Error correction - Identification (control systems) - Image compression - Neural networks - Pattern recognition systems - Speech recognition - Visual communication
Uncontrolled terms: Automatic modulation recognition - Automatic voltage regulators (AVR) - Autonomous wind diesel system - Data glove devices - EiRev - Energy miser scheme - Software package MATLAB - Software package SIMULINK - Static random access memory (SRAM) - Three phase power transformers
Classification Code: 717.1 Optical Communication Systems - 722.3 Data Communication, Equipment and Techniques - 723.4 Artificial Intelligence - 723.5 Computer Applications - 731.1 Control Systems - 751.5 Speech
Treatment: Applications (APP) - General review (GEN) - Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

97. Computational studies of flow through cross flow fans - Effect of blade geometry

Govardhan, M. (1); Sampat, D. Lakshmana (2)

Source: *Journal of Thermal Science*, v 14, n 3, p 220-229, September 2005; **ISSN:** 10032169; **Publisher:** Editorial Committee of Journal of Thermal Sciences
Author affiliation: (1) School of Mechanical Engineering, Universiti Malaysia Sabah, Sabah, Malaysia (2) Thermal Turbomachines Laboratory, Department of Mechanical Engineering, Indian Institute of Technology, Madras, Chennai, India
Abstract: This present paper describes three dimensional computational analysis of complex internal flow in a cross flow fan. A commercial computational fluid dynamics (CFD) software code CFX was used for the computation. RNG k-ε two equation turbulence model was used to simulate the model with unstructured mesh. Sliding mesh interface was used at the interface between the rotating and stationary domains to capture the unsteady interactions. An accurate assessment of the present investigation is made by comparing various parameters with the available experimental data. Three impeller geometries with different blade angles and radius ratio are used in the present study. Maximum energy transfer through the impeller takes place in the region where the flow follows the blade curvature. Radial velocity is not uniform through blade channels. Some blades work in turbine mode at very low flow coefficients. Static pressure is always negative in and around the impeller region. (16 refs)
Main heading: Flow interactions
Controlled terms: Computational fluid dynamics - Energy transfer - Fans - Flow patterns - Flow visualization - Impellers - Steady flow - Turbomachine blades - Turbulent flow
Uncontrolled terms: Blade geometry - Complex internal flow - Cross flow fans - Multiple frames of reference - Quasi steady state
Classification Code: 601.2 Machine Components - 618.3 Blowers and Fans - 631.1 Fluid Flow, General - 723.5 Computer Applications
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

98. 2009 IEEE 17th International Conference on Program Comprehension, ICPC '09

Source: *IEEE International Conference on Program Comprehension*, 2009, *2009 IEEE 17th International Conference on Program Comprehension, ICPC '09*; **ISBN-13:** 9781424439973; **Conference:** 2009 IEEE 17th International Conference on Program Comprehension, ICPC '09, May 17, 2009 - May 19, 2009; **Publisher:** IEEE Computer Society
Abstract: The proceedings contain 56 papers. The topics discussed include: intensions are a key to program comprehension; variable granularity for improving precision of impact analysis; automatically identifying changes that impact code-to-design traceability; automatic classification of large changes into maintenance categories; practical static analysis for inference of security-related program properties; impact analysis and visualization toolkit for static crosscutting in AspectJ; BugFix: a learning-based tool to assist developers in fixing bugs; resumption strategies for interrupted programming tasks; Using activity traces to characterize programming behavior beyond the lab; an in-vivo study of the cognitive levels employed by programmers during software maintenance; and trace visualization for program comprehension: a controlled experiment.
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

99. 2013 International Forum on Mechanical and Material Engineering, IFMME 2013

Source: *Advanced Materials Research*, v 774-776, 2013, *Advanced Technologies in Manufacturing, Engineering and Materials*; **ISSN:** 10226680; **ISBN-13:** 9783037858004; **Conference:** 2013 International Forum on Mechanical and Material Engineering, IFMME 2013, June 13, 2013 - June 14, 2013; **Sponsor:** Korea Maritime University; Inha University; Hong Kong Industrial Technology Research Centre; **Publisher:** Trans Tech Publications Ltd
Abstract: The proceedings contain 406 papers. The special focus in this conference is on Dynamic Systems, Vibration and Noise, Applied Mechanics, Design and Modelling in Manufacture, Dynamic Simulation, Machinery and Equipments, Fluid, Flow Engineering and Control Technology, Aerodynamics, Wind and Heat Engineering, Vehicle Engineering, Material Science and Technology, Material Science and Technology, Advanced Manufacturing Technology and Mechatronics, Control Technology and Automation Systems and Communication and Signal Engineering. The topics include: topology optimization of passive constrained layer damping on plates with respect to noise control; structure simulation analysis of piezoelectric energy harvesting device; SRSM-based stochastic model updating method; study on surface radiation noise of a diesel engine based on FEM; vibration simulations of double-walled carbon nanotubes by finite element method; finite element analysis of tandem rubber o-ring sealing structure at the end of shaft; simulation and test for nonlinear dynamic specialty of rubber mounting; synthetical dynamic balancing of mechanism based on optimization simulation; study on self-excited vibration of subsoiler-soil system based on

modal analysis; the research of damping adjustable isolators control method; study on the mechanical properties of magnetorheological elastomers; analysis of the movement trajectory of blasting flyrock in the bottom of open pits; research on aerodynamic impacts of snow on bridge decks; the vibration and modal analysis of the disc brake; the attenuation trend of the propeller noise; the Geneva mechanism motion simulation and research to improve the design method; optimization design of locking mechanisms of small arms; design of cigarette universal code printing system; fatigue strength check of flexspline with inner and outer gear ring of the micro harmonic drive; study on the gear tooth influence coefficients of flexspline of harmonic drive; design of an extrusion and heat setting machine for needled filter felts; finite element analysis for critical components of ultra-precision grinding machine tool; optimization design of planetary gear transmission mechanism of traction unit; design and optimization of magnetic levitation actuators for active vibration isolation system; supplement of several problems in roller chain drive design; research on case similarity method of spade punch planter; research on reverse engineering for blades of axial compressors with laser scanner; design and calculation of the cam mechanism profile curve based on MATLAB; dynamic simulation of small crawler chassis turning based on RecurDyn; simulation and analysis of rail eddy current brake system of high-speed train; new design of supercritical water oxidation reactor for sewage sludge treatment; thermal analysis of CMT structure; use and maintenance of CNC machine tools; the design of hydraulic vibratory fruit-vine separation system for processing tomato; simulation study on biomechanics of knee joint in running; kinematics simulation study on a manual operation sugarcane loader with metamorphic function; numerical simulation on aerodynamic characteristics of road vehicles on bridges under cross winds; parameterized modeling and analysis of wind turbine blade using VB and ANSYS; numerical simulation of three-dimensional unsteady flow field in the cyclone; analysing for flow field of water displacing oil in microscopic pore; wind tunnel test of honeycomb in improving flow quality; study on electromagnetic vibration pump; numerical calculation of fluid flow in a continuous casting tundish; two new flow control technologies based on vibration; application of Bernoulli equation in fluid mechanics; flameless venting devices for dust explosions and experimental confirmation of their efficiency; 3D wave simulation basing on VOF method and dynamic grid technology; effects of liquid properties on acoustic gas bubble radial oscillation; design of pneumatic transmission system base on FSC car of south china university of technology; characteristics analysis of hydraulic system for screw distributor of asphalt paver; prediction of remaining useful life for used gas turbine blades; mathematical simulation the influence of tundish with a retaining wall; flow behavior of polymer viscoelastic fluid in complex channel; the effects of parameters on HRSG thermodynamic performance; determination of load capacity and deformation for modular trailer; design and practice of FSAE car dynamic data acquisition system; design of bus air-conditioning control system based on PIC18F458; technology of vehicle multisystem energy-harvest absorber; study on driving safety of refueling truck; a research on modular training system of automobile engine; hardware in the loop platform with brake-by-wire and steer-by-wire system; vehicle stability control based on generalized predictive contol; transmission ratio research of hydraulic steering by-wire system; unloading behavior study on FRP-concrete interface under an exponential softening law; pyrolysis features of larch bark and xylem; electrochemical study on the bioleaching of marmatite; the mechanical performance degradation of steels effected by corrosion; magnetic states of nanoparticles with RKKY interaction; tests on effect of gravity and soil density on soil cone index; preparation and characterization of magnetic graphene oxide; immobilization of ?-glucosidase on modified attapulgite; failure mechanisms of nanoparticle reinforced metal matrix composite; microwave pre-treatment reduced the browning of peach juice; a facile approach to synthesis and modification of nano-alumina; study of nanoparticle iron-nitride MRF and its rheological characteristic; thermogravimetric analysis of the reduction of iron ore with hydroxyl content; preparation of tungsten oxide nanoplate thin film and its gas sensing properties; research progress of quantum dot intermediate band solar cell; mobile adverting: the new promising marketing channel; analysis of Chinese CNC machine tool industry based on patent intelligence; seamless safe supervision system of edible agricultural products; research on business process management system based on SOA; airline safety risk evaluation based on factor analysis; the design of smart alcohol testing system based on internet of things; computational trust in cloud manufacturing; a healthcare service system based on internet of things; building information architecture of the internet of things; optimization models based on line loss post management in power system; a new method to optimize locations of svc based on risk and static load margin; multitask fuzzy learning with rule weight; simulation of nonlinear transformer on open and short circuit; optimal reactive power compensation in power system considering SVC; tollerance coefficients of stability in linear dynamic system calculated in matlab environment; an algorithm to detect noised pixel in image; the visualization of soil moisture based on VB and surfer; a portable weather station design applied in the smart city; grey-hierarchy relation analysis and evaluation of port operating ability; comparison between normalized databases implemented with different database systems; the exponent set of a class of two-colored digraphs with one common vertex; CUDA parallel computing combined with OpenGL interoperate; the processing and analyzing of non-structured data in digital investigation; a web information extraction method based on HTML parser; study on knowledge services based on multi-media data mining; research on the construction of service mode in university mobile digital library; research and application of OPC network communication in configuration software; research on the initial value of the simulated annealing; a system hierarchical method based on self-resolving message protocol; using the pairing function for distributed access control in cloud computing; a user's profile model for cloud computing service; study of dynamic spectrum access scheme in HD radio; RFID technology-based the railway package management platform design;

residential extraction based on joint feature of spectrum and spatial; an automatic barn-entering system for grains; software system development of crane structural health monitoring; multi-frequency heterodyne phase shift technology in 3-D measurement; obstacle detection of autonomous land vehicles for geological prospecting; an improved fast algorithm of spacecraft separation distance calculation; a survey on coverage control technology in wireless sensor network; a novel image fusion based on nonsubsampled contourlet transform; feature subset selection based on the genetic algorithm; the design of PID control system which based on the STM32; automated feeding system for meat pigeon based on STM32; study on speed control method; computer numerical control machine tool and its development trend; the introduction of COREX process development; application and development of ship welding material; microwave PDC drill bit; study of low-load industrial robot; autonomous navigation system based on GPS for agricultural vehicles; a balancing mechanism for an anthropomorphic robot; forecasting the success of implementing advanced manufacturing technology; manufacturing resource ontology modeling techniques based on carrier; research on dynamics of a three-DOF parallel manipulator for levelling mechanism; improving of manufacturing systems in Slovak industrial enterprises; laser-ultrasound testing of cracks in porcelain insulator; design of artificial hip joint by carbon/PEEK composites; design of a testing system for strength of ceramic sintering; the manufacture of spherical WC powder in plasma; a method of evaluating carton black's morphology based on convex hull; a method of evaluating carton black's dispersion based on standard images and study on urea-formaldehyde resin for glass wool products.
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

100. A macroblock-based perceptually adaptive bit allocation for H264 rate control

Hrarti, Miryem (1, 2); Saadane, Hakim (2); Larabi, Mohamed-Chaker (2); Tamtaoui, Ahmed (3); Aboutajdine, Driss (1)
Source: *2010 5th International Symposium on I/V Communications and Mobile Networks, ISIVC 2010*, 2010, *2010 5th International Symposium on I/V Communications and Mobile Networks, ISIVC 2010*; **ISBN-13:** 9781424459988; **DOI:** 10.1109/ISVC.2010.5656263; **Article number:** 5656263; **Conference:** 2010 5th International Symposium on I/V Communications and Mobile Networks, ISIVC 2010, September 30, 2010 - October 2, 2010; **Sponsor:** Islam. Educ., Sci. Cult. Organ. (ISESCO); National Scientific and Technical Research Centre (CNRST); Minist. Natl. Educ. Higher Educ., Prof. Train. Sci. Res.; Service for Cooperation and Cultural Action; Pole of Competences STIC; **Publisher:** IEEE Computer Society
Author affiliation: (1) GSCM-Lrit Laboratory Attached to CNRST, Faculty of Sciences, University Mohamed, Morocco (2) XLim Institute, Department of Signal, Image and Communication, University of Poitiers, France (3) LTI, National Institute of Post and Telecommunications (INPT), Rabat, Morocco
Abstract: Statistical methodologies are the main tools used in video compression and this lead to a kind of stagnation in terms of performance. This means that solutions for increasing the visual performance of video compression have to come from other fields like the perception. One can notice that coding errors in highly textured areas are relatively less perceptible than errors in untextured ones because of the masking effect. The existing H.264/AVC bit allocation scheme does not take into account this phenomenon. To handle this problem, we propose an adaptive bit allocation based on spatial and temporal perceptual features. This is performed by determining a spatiotemporal importance factor that is used to adjust the number of allocated bits. The proposed spatial feature consists on classifying regions into three categories: flat, textured and edged regions. The proposed temporal feature assumes that the Human Visual System (HVS) is more sensitive for moving regions than static ones. So, a low amount of bits will be assigned to static textured regions, and large one will be assigned to moving regions, which are spatially edged or flat. Experimental results show that the proposed bit allocation at macroblock (MB) level, compared with H.264/AVC reference software, improves the average peak signal-to-noise ratio (PSNR) (up to +1.10dB) and preserves details in the most perceptually prominent regions for low bitrates. © 2010 IEEE. (17 refs)
Main heading: Image coding
Controlled terms: Coding errors - Electric distortion - Image compression - Image quality - Motion estimation - Motion Picture Experts Group standards - Optimization - Signal distortion - Signal to noise ratio - Statistical methods - Textures - Video signal processing - Wireless networks
Uncontrolled terms: Bit allocation - H.264/AVC - Human Visual System - Rate-distortion optimization - Temporal classification - Texture analysis
Classification Code: 933 Solid State Physics - 922.2 Mathematical Statistics - 921.5 Optimization Techniques - 741 Light, Optics and Optical Devices - 723.2 Data Processing and Image Processing - 716 Telecommunication; Radar, Radio and Television - 711.1 Electromagnetic Waves in Different Media
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village