1. Visualization and evolution of software architectures

Khan, Taimur (1); Barthel, Henning (2); Ebert, Achim (1); Liggesmeyer, Peter (2)
Source: *OpenAccess Series in Informatics*, v 27, p 25-42, 2012, *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering, VLUDS 2011 - Proceedings of IRTG 1131 Workshop*; **ISSN:** 21906807; **ISBN-13:** 9783939897460; **DOI:** 10.4230/OASIcs.VLUDS.2011.25; **Conference:** International Research and Training Group 1131 Workshop on Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering, VLUDS 2011, June 10, 2011 - June 11, 2011; **Publisher:** Schloss Dagstuhl - Leibniz-Zentrum fur Informatik GmbH,
Author affiliation: (1) Computer Graphics and HCI Group, University of Kaiserslautern, Germany (2) Fraunhofer IESE, Kaiserslautern, Germany
Abstract: Software systems are an integral component of our everyday life as we find them in tools and embedded in equipment all around us. In order to ensure smooth, predictable, and accurate operation of these systems, it is crucial to produce and maintain systems that are highly reliable. A well-designed and well-maintained architecture goes a long way in achieving this goal. However, due to the intangible and often complex nature of software architecture, this task can be quite complicated. The field of software architecture visualization aims to ease this task by providing tools and techniques to examine the hierarchy, relationship, evolution, and quality of architecture components. In this paper, we present a discourse on the state of the art of software architecture visualization techniques. Further, we highlight the importance of developing solutions tailored to meet the needs and requirements of the stakeholders involved in the analysis process. (78 refs)
Main heading: Software architecture
Controlled terms: Computer software maintenance - Visualization
Uncontrolled terms: Architecture visualization - Developing solutions - Human perception - Integral components - Software comprehension - Software Evolution - State of the art - Tools and techniques
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics
Database: Compendex

2. Software evolution: Analysis and visualization

Gall, Harald C. (1); Lanza, Michele (2)
Source: *Proceedings - International Conference on Software Engineering*, v 2006, p 1055-1056, 2006, *Proceeding of the 28th International Conference on Software Engineering 2006, ICSE '06*; **ISSN:** 02705257; **ISBN-10:** 1595933751, **ISBN-13:** 9781595933751; **Conference:** 28th International Conference on Software Engineering 2006, ICSE '06, May 20, 2006 - May 28, 2006; **Sponsor:** ACM Special Interest Group on Software Engineering, SIGSOFT; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) Department of Informatics, University of Zurich (2) Faculty of Informatics, University of Lugano, Switzerland
Abstract: Gaining higher level evolutionary information about large software systems is a key challenge in dealing with increasing complexity and decreasing software quality. Software repositories such as modifications, changes, or release information are rich sources for distinctive kinds of analyses: They reflect the reasons and effects of particular changes made to the software system over a certain period of time. If we can analyze these repositories in an effective way, we get a clearer picture of the status of the software. Software repositories can be analyzed to provide information about the problems concerning a particular feature or a set of features. Hidden dependencies of structurally unrelated but over time logically coupled files exhibit a high potential to illustrate software evolution and possible architectural deterioration. In this tutorial, we describe the investigation of software evolution by taking a step towards reflecting the analysis results against software quality attributes. Different kinds of analyses (from architecture to code) and their interpretation will be presented and discussed in relation to quality attributes. This will show our vision of where such evolution investigations can lead and how they can support development. For that, the tutorial will touch issues such as meta-models for evolution data, data analysis and history mining, software quality attributes, as well as visualization of analysis results. (14 refs)
Main heading: Software engineering
Controlled terms: Computational complexity - Data mining - Data reduction - Quality of service
Uncontrolled terms: History mining - Software evolution - Software quality
Classification Code: 716 Telecommunication; Radar, Radio and Television - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 723.1 Computer Programming - 723.2 Data Processing and Image Processing
Treatment: Theoretical (THR)
Database: Compendex

Data Provider: Engineering Village

3. Ontology-driven visualization of architectural design decisions

De Boer, Remco C. (1); Lago, Patricia (1); Telea, Alexandru (2); Van Vliet, Hans (1)
Source: *2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA 2009*, p 51-60, 2009, *2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA 2009*; **ISBN-13:** 9781424449859; **DOI:** 10.1109/WICSA.2009.5290791; **Article number:** 5290791; **Conference:** 2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA 2009, September 14, 2009 - September 17, 2009; **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computer Science, VU University Amsterdam, Netherlands (2) Institute for Mathematics and Computer Science, University of Groningen, Netherlands
Abstract: There is a gradual increase of interest to use ontologies to capture architectural knowledge, in particular architectural design decisions. While ontologies seem a viable approach to codification, the application of such codified knowledge to everyday practice may be non-trivial. In particular, browsing and searching an architectural knowledge repository for effective reuse can be cumbersome. In this paper, we present how ontology-driven visualization of architectural design decisions can be used to assist software product audits, in which independent auditors perform an assessment of a product's quality. Our visualization combines the simplicity of tabular information representation with the power of on-the-fly ontological inference of decision attributes typically used by auditors. In this way, we are able to support the auditors in effectively reusing their know-how, and to actively assist the core aspects of their decision making process, namely trade-off analysis, impact analysis, and if-then scenarios. We demonstrate our visualization with examples from a real-world application. © 2009 IEEE. (18 refs)
Main heading: Software architecture
Controlled terms: Architectural design - Computer software - Decision making - Ontology - Structural design - Technology transfer - Visualization
Uncontrolled terms: Architectural knowledge - Decision attribute - Decision making process - Impact analysis - Information representation - Know-how - Non-trivial - On-the-fly - Real-world application - Software products - Trade-off analysis
Classification Code: 911.2 Industrial Economics - 903 Information Science - 902.1 Engineering Graphics - 901.4 Impact of Technology on Society - 912.2 Management - 723.5 Computer Applications - 723 Computer Software, Data Handling and Applications - 408.1 Structural Design, General - 402 Buildings and Towers - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

4. On quick comprehension and assessment of software

Bartoszuk, Cezary (1); Dabrowski, Robert (1); Stencel, Krzysztof (1); Timoszuk, Grzegorz (1)
Source: *ACM International Conference Proceeding Series*, v 767, p 161-168, 2013, *Computer Systems and Technologies: 14th International Conference, CompSysTech 2013 - Proceedings*; **ISBN-13:** 9781450320214; **DOI:** 10.1145/2516775.2516806; **Conference:** 14th International Conference on Computer Systems and Technologies, CompSysTech 2013, June 28, 2013 - June 29, 2013; **Sponsor:** Technical University of Varna, Bulgaria (TECHUVB); Federation of the Scientific Eng. Unions - Bulgaria (FOSEUB); Bulgarian Ministry of Education, Youth and Science (MEYS); CASTUVTB; University of Ruse, Bulgaria (UORB); **Publisher:** Association for Computing Machinery
Author affiliation: (1) Institute of Informatics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
Abstract: By an architecture of a software system we mean the fundamental organization of the system embodied in its components, their relationships to one another and to the system's environment. It also encompasses principles governing the system's design and evolution. Architectures of complex systems are obviously complex as well.
The goal of our research is to harness this complexity. In this paper we focus on providing software architects with ability to quickly comprehend the complexity and assess the quality of software. The essential tools we use are: (1) a graph-based repository for collecting information on software artefacts, accompanied by (2) tools to perform software intelligence tasks, like analyzing dependencies among those artefacts, calculating their importance, and quality. On top of those tools we implement visualization methods that render the relative importance using size and the quality using colours. By means of such methods a software architect can at glance comprehend and assess the software, He/she can (1) find the starting points to dig into a complex system; (2) judge the cohesion and coupling of system components; and (3) assess the overall quality. We demonstrate this method using selected open-source projects of various sizes and qualities. © 2013 ACM. (26 refs)
Main heading: Tools

Controlled terms: Architecture - Computer software - Image quality - Large scale systems - Software architecture
Uncontrolled terms: Cohesion and couplings - graph - intelligence - metrics - Open source projects - Quality of softwares - Software intelligences - Visualization method
Classification Code: 402 Buildings and Towers - 603 Machine Tools - 605 Small Tools and Hardware - 723 Computer Software, Data Handling and Applications - 741 Light, Optics and Optical Devices - 961 Systems Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

5. Towards quantitative evaluation of UML based software architecture

Li, Jinhua (1); Guo, Zhenbo (1); Zhao, Yun (1); Zhang, Zhenhua (1); Pang, Ruijuan (1)
Source: *Proceedings - SNPD 2007: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, v 1, p 663-669, 2007, *Proceedings - SNPD 2007: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*; **ISBN-10:** 0769529097, **ISBN-13:** 9780769529097; **DOI:** 10.1109/SNPD.2007.551; **Article number:** 4287589; **Conference:** SNPD 2007: 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, July 30, 2007 - August 1, 2007; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) College of Information Engineering, Qingdao University
Abstract: The architecture of a software system is a critical artifact in the software lifecycle and should be evaluated as early as possible. Recent efforts to software architecture evaluation are concentrated on scenario-based methods which are qualitative, subjective and need not any special architecture description languages. This paper investigates an approach to metrics based quantitative evaluation of UML software architecture. UML is a visual modeling language with well-formed hierarchical syntax and semantics, and is uniformly applied in various development stages. With supplementation UML has been adapted to describing software architecture. By utilization of these features three types of metrics for UML diagrams are proposed They measure the amount of information, visual effect and connectivity degree in different UML diagrams. The application of these metrics in quantitative evaluating qualities at the architecture-level such as system scale, complexity and structural characteristics is discussed. © 2007 IEEE. (17 refs)
Main heading: Unified Modeling Language
Controlled terms: Computational complexity - Feature extraction - Hierarchical systems - Software architecture - Syntactics
Uncontrolled terms: Architecture description languages - Hierarchical syntax - Software lifecycle - System scales
Classification Code: 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 723.1.1 Computer Programming Languages - 723.5 Computer Applications - 903.2 Information Dissemination - 961 Systems Science
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

6. Preventing erosion of architectural tactics through their strategic implementation, preservation, and visualization

Mirakhorli, Mehdi (1)
Source: *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings*, p 762-765, 2013, *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings*; **ISBN-13:** 9781479902156; **DOI:** 10.1109/ASE.2013.6693152; **Article number:** 6693152; **Conference:** 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, November 11, 2013 - November 15, 2013; **Sponsor:** IEEE Computer Society; Association for Computing Machinery, Special Interest Group on Software Engineering (ACM SIGSOFT); IEEE Technical Council on Software Engineering (TCSE); ACM SIGART; NASA; **Publisher:** IEEE Computer Society
Author affiliation: (1) DePaul University, School of Computing, Chicago, IL 60604, United States
Abstract: Nowadays, a successful software production is increasingly dependent on how the final deployed system addresses customers' and users' quality concerns such as security, reliability, availability, interoperability, performance and many other types of such requirements. In order to satisfy such quality concerns, software architects are accountable for devising and comparing various alternate solutions, assessing the trade-offs, and finally adopting strategic design decisions which optimize the degree to which each of the quality concerns is satisfied. Although designing and implementing a good architecture is necessary, it is not usually enough. Even a good architecture can

deteriorate in subsequent releases and then fail to address those concerns for which it was initially designed. In this work, we present a novel traceability approach for automating the construction of traceabilty links for architectural tactics and utilizing those links to implement a change impact analysis infrastructure to mitigate the problem of architecture degradation. Our approach utilizes machine learning methods to detect tactic-related classes. The detected tactic-related classes are then mapped to a Tactic Traceability Pattern. We train our trace algorithm using code extracted from fifty performance-centric and safety-critical open source software systems and then evaluate it against a real case study. © 2013 IEEE. (25 refs)
Main heading: Learning systems
Controlled terms: Architecture - Interoperability - Software architecture - Software reliability
Uncontrolled terms: Change impact analysis - Machine learning methods - Open source software systems - Software architects - Software production - tactics - traceability - traceability patterns
Classification Code: 402 Buildings and Towers - 716 Telecommunication; Radar, Radio and Television - 717 Optical Communication - 718 Telephone Systems and Related Technologies; Line Communications - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

7. A tool to visualize architectural design decisions

Lee, Larix (1); Kruchten, Philippe (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 5281 LNCS, p 43-54, 2008, *Quality of Software Architectures: Models and Architectures - 4th International Conference on the Quality of Software Architectures, QoSA 2008, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3540878785, **ISBN-13:** 9783540878780; **DOI:** 10.1007/978-3-540-87879-7-3; **Conference:** 4th International Conference on the Quality of Software Architectures, QoSA 2008, October 14, 2008 - October 17, 2008; **Publisher:** Springer Verlag
Author affiliation: (1) University of British Columbia
Abstract: The software architecture community is shifting its attention to architectural design decisions as a key element of architectural knowledge. Although there has been much work dealing with the representation of design decisions as formal structures within architecture, there still remains a need to investigate the exploratory nature of the design decisions themselves. We present in this paper a tool that should help improve the quality of software architecture by enabling design decision exploration and analysis through decision visualization. Unlike many other design decision tools which acquire, list, and perform queries on decisions, our tool provides visualization components to help with decision exploration and analysis. Our tool has four main aspects: 1) the decision and relationship lists; 2) decision structure visualization view; 3) decision chronology view; and 4) decision impact view. Together, these four aspects provide an effective and powerful means for decision exploration and analysis. © 2008 Springer Berlin Heidelberg. (24 refs)
Main heading: Software architecture
Controlled terms: Architectural design - Computer software selection and evaluation - Visualization
Uncontrolled terms: Architectural knowledge - Decision impacts - Design decisions - Design-decision tools - Key elements - Quality of softwares
Classification Code: 912.2 Management - 902.1 Engineering Graphics - 723.5 Computer Applications - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 408.1 Structural Design, General - 402 Buildings and Towers
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

8. A visual analysis and design tool for planning software reengineerings

Beck, Martin (1); Trümper, Jonas (1); Döllner, Jürgen (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.2011.6069458; **Article number:** 6069458; **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) Hasso-Plattner-Institute, University of Potsdam, Germany

Abstract: Reengineering complex software systems represents a non-trivial process. As a fundamental technique in software engineering, reengineering includes (a) reverse engineering the as-is system design, (b) identifying a set of transformations to the design, and (c) applying these transformations. While methods a) and c) are widely supported by existing tools, identifying possible transformations to improve architectural quality is not well supported and, therefore, becomes increasingly complex in aged and large software systems. In this paper we present a novel visual analysis and design tool to support software architects during reengineering tasks in identifying a given software's design and in visually planning quality-improving changes to its design. The tool eases estimating effort and change impact of a planned reengineering. A prototype implementation shows the proposed technique's feasibility. Three case studies conducted on industrial software systems demonstrate usage and scalability of our approach. © 2011 IEEE. (37 refs)
Main heading: Software design
Controlled terms: C (programming language) - Computer software - Design - Quality control - Reengineering - Reverse engineering - Software architecture - Systems analysis
Uncontrolled terms: Architectural quality - Complex software systems - Industrial software - Large software systems - Non-trivial - Prototype implementations - Software architects - Visual analysis
Classification Code: 408 Structural Design - 723 Computer Software, Data Handling and Applications - 913.3 Quality Assurance and Control - 961 Systems Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

9. Applying source code analysis techniques: A case study for a large mission-critical software system

Haralambiev, Haralambi (1); Boychev, Stanimir (1); Lilov, Delyan (1); Kraichev, Kraicho (1)
Source: *EUROCON 2011 - International Conference on Computer as a Tool - Joint with Conftele 2011*, 2011, *EUROCON 2011 - International Conference on Computer as a Tool - Joint with Conftele 2011*; **ISBN-13:** 9781424474868; **DOI:** 10.1109/EUROCON.2011.5929241; **Article number:** 5929241; **Conference:** International Conference on Computer as a Tool, EUROCON 2011 - Joint with Conftele 2011, April 27, 2011 - April 29, 2011; **Sponsor:** Autoridade Nacional de Comunicacoes (ANACOM); LOGISER; **Publisher:** IEEE Computer Society
Author affiliation: (1) Applied Research and Development Center, MuSala Soft, Sofia, Bulgaria
Abstract: Source code analysis has been and still is extensively researched topic with various applications to the modern software industry. In this paper we share our experience in applying various source code analysis techniques for assessing the quality of and detecting potential defects in a large mission-critical software system. The case study is about the maintenance of a software system of a Bulgarian government agency. The system has been developed by a third-Party software vendor over a period of four years. The development produced over 4 million LOC using more than 20 technologies. MuSala Soft won a tender for maintaining this system in 2008. Although the system was operational, there were various issues that were known to its users. So, a decision was made to assess the system's quality with various source code analysis tools. The expectation was that the findings will reveal some of the problems' cause, allowing us to correct the issues and thus improve the quality and focus on functional enhancements. MuSala Soft had already established a special unit Applied Research and Development Center dealing with research and advancements in the area of software system analysis. Thus, a natural next step was for this unit to use the know-how and in-house developed tools to do the assessment. The team used various techniques that had been subject to intense research, more precisely: software metrics, code clone detection, defect and code smells detection through flow-sensitive and points-to analysis, software visualization and graph drawing. In addition to the open-source and free commercial tools, the team used internally developed ones that complement or improve what was available. The internally developed Smart Source Analyzer platform that was used is focused on several analysis areas: source code modeling, allowing easy navigation through the code elements and relations for different programming languages; quality audit through software metrics by aggregating various metrics into a more meaningful quality characteristic (e.g. "maintainability"); source code pattern recognition to detect various security issues and "code smells". The produced results presented information about both the structure of the system and its quality. As the analysis was executed in the beginning of the maintenance tenure, it was vital for the team members to quickly grasp the architecture and the business logic. On the other hand, it was important to review the detected quality problems as this guided the team to quick solutions for the existing issues and also highlighted areas that would impede future improvements. The tool IPlasma and its System Complexity View (Fig. 1) revealed where the business logic is concentrated, which are the most important and which are the most complex elements of the system. The analysis with our internal metrics framework (Fig. 2) pointed out places that need refactoring because the code is hard to modify on request or testing is practically impossible. The code clone detection tools showed places where copy and paste programming has been applied. PMD, Find Bugs and Klockwork Solo tools were used to detect various "code smells" (Fig. 3). There were a number of occurrences that were indeed bugs in the system. Although these results were productive for the successful execution of the project, there were some challenges that should be addressed in the future through more

extensive research. The two aspects we consider the most important are uSability and integration. As most of the tools require very deep understanding of the underlying analysis, the whole process requires tight cooperation between the analysis team and the maintenance team. For example, most of the metrics tools available provide specific values for a given metric without any indication what the value means and what is the threshold. Our internal metrics framework aggregates the metrics into meaningful quality characteristics, which solves the issue Partially. However, the user still often wonders about the justification behind the meaning of the given quality characteristic. There is a need for an explanation system one, which could point out the source code elements and explain why they are considered good or bad. The integration aspect is considered important because such analysis should be performed continuously. In our experience, the analysis is usually performed subsequent to an important event in this case: beginning of maintenance tenure. Some quality assurance practices should be developed and then adopted by the development teams so that the implementation quality is checked continuously. This should cover various activities and instruments, such as the integrated development environment, the code review process, automated builds, etc. In conclusion, we think that implementation quality audit and management is a vital activity that should be integrated into the software development process and the tools that support it should be uSable by the development team members without much knowledge of the underlying analysis. In this paper we presented a case study that showed the benefits of such a process. © 2011 IEEE.
Main heading: Quality control
Controlled terms: Cloning - Codes (symbols) - Computer software - Computer software maintenance - Defects - Drawing (graphics) - Equipment - Integration - Maintainability - Object oriented programming - Odors - Pattern recognition - Problem oriented languages - Program debugging - Quality assurance - Research - Software design - Systems analysis - Technology transfer - Visualization - Web services
Uncontrolled terms: Applied research - Business logic - Code clone detection - Code review - Code smell - Commercial tools - Copy-and-paste programming - Development teams - Explanation systems - Functional enhancements - Government agencies - Graph drawing - implementation quality - Integrated development environment - Know-how - Mission critical softwares - Open-source - Points-to analysis - Potential defects - Quality assurance practices - Quality characteristic - Quality problems - Refactorings - Security issues - Software development process - Software industry - software metrics - Software system analysis - Software systems - Software vendors - Software visualization - Source code analysis - Source codes - Specific values - System complexity - System's quality - Team members - Whole process
Classification Code: 951 Materials Science - 921.2 Calculus - 913.5 Maintenance - 913.3 Quality Assurance and Control - 902.1 Engineering Graphics - 961 Systems Science - 901 Engineering Profession - 716 Telecommunication; Radar, Radio and Television - 461.8.1 Genetic Engineering - 451.1 Air Pollution Sources - 423 Non Mechanical Properties and Tests of Building Materials - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

10. Code rocket: Improving detailed design support in mainstream software development

Parkes, Steve (1); Ramsay, Craig (1); Spark, Alan (2)
Source: *2011 International Conference on Computer and Management, CAMAN 2011*, 2011, *2011 International Conference on Computer and Management, CAMAN 2011*; **ISBN-13:** 9781424492831; **DOI:** 10.1109/CAMAN.2011.5778773; **Article number:** 5778773; **Conference:** 2011 International Conference on Computer and Management, CAMAN 2011, May 19, 2011 - May 21, 2011; **Sponsor:** IEEE Wuhan Section; Hunan University; Wuhan University; Engineering Information Institute; Chongqing VIP Information Co., Ltd; **Publisher:** IEEE Computer Society
Author affiliation: (1) School of Computing, University of Dundee, Dundee, United Kingdom (2) Research and Development, Rapid Quality Systems Ltd., Dundee, United Kingdom
Abstract: In mainstream software development there can often be a gap which exists between the stages of performing the architectural design of a software system and implementing the detailed algorithms and processes required in the program code. Code Rocket is a code visualization and documentation system which has been developed to fill this gap. Code Rocket provides automated design and documentation support for software developers during detailed stages of code construction. It integrates seamlessly with existing development tools to provide extensive documentation with little or no effort on behalf of the software engineer. Code and documentation remain fully synchronized even when changes are implemented in the code. This paper describes Code Rocket, the rationale for its development, and the key features and benefits it delivers to different stakeholders on a software project. ©2011 IEEE. (12 refs)
Main heading: Software design
Controlled terms: Architectural design - Computer software - Rockets - Visualization

Uncontrolled terms: Automated design - Code construction - Code visualization - Design tools and techniques - Detailed design - Development tools - Documentation systems - Key feature - Program code - Software developer - Software engineers - Software project - Software systems
Classification Code: 402 Buildings and Towers - 404.1 Military Engineering - 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

11. Is it possible to decorate graphical software design and architecture models with qualitative information? - An experiment

Bratthall, Lars (1); Wohlin, Claes (2)
Source: *IEEE Transactions on Software Engineering*, v 28, n 12, p 1181-1193, December 2002; **ISSN:** 00985589; **DOI:** 10.1109/TSE.2002.1158290; **Publisher:** Institute of Electrical and Electronics Engineers Inc.
Author affiliation: (1) Corporate Research Department, ABB AS, Norway, Bergervn 12, N-1375 Billingstad, Norway
(2) Dept. of Software Eng./Comp. Sci., Blekinge Institute of Technology, PO Box 520, SE-372 25 Ronneby, Sweden
Abstract: Software systems evolve over time and it is often difficult to maintain them. One reason for this is that often it is hard to understand the previous release. Further, even if architecture and design models are available and up to date, they primarily represent the functional behavior of the system. To evaluate whether it is possible to also represent some nonfunctional aspects, an experiment has been conducted. The objective of the experiment is to evaluate the cognitive suitability of some visual representations that can be used to represent a control relation, software component size and component external and internal complexity. Ten different representations are evaluated in a controlled environment using 35 subjects. The results from the experiment show that representations with low cognitive accessibility weight can be found. In an example, these representations are used to illustrate some qualities in an SDL block diagram. It is concluded that the incorporation of these representations in architecture and design descriptions is both easy and probably worthwhile. The incorporation of the representations should enhance the understanding of previous releases and, hence, help software developers in evolving and maintaining complex software systems. (44 refs)
Main heading: Software engineering
Controlled terms: Computational complexity - Computer software maintenance - Mathematical models - Statistical methods
Uncontrolled terms: Software evolution
Classification Code: 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 723.1 Computer Programming - 921 Mathematics - 922.2 Mathematical Statistics
Treatment: Theoretical (THR) - Experimental (EXP)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

12. Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse

Lintern, Rob (1); Michaud, Jeff (1); Storey, Margaret-Anne (1); Wu, Xiaomin (1)
Source: *Proceedings of ACM Symposium on Software Visualization*, p 47-56, 2003, *Proceedings of the 2003 ACM Symposium on Software Visualization, SoftVis '03*; **ISBN-10:** 1581136420, **ISBN-13:** 9781581136425; **DOI:** 10.1145/774833.774840; **Conference:** Proceedings of the ACM 2003 Symposium on Software Visualization (SoftVis 2003), June 11, 2003 - June 13, 2003; **Sponsor:** ACM SIGCHI; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Dept. of Computer Science, University of Victoria, Victoria, BC, Canada
Abstract: The Eclipse platform presents an opportunity to openly collaborate and share visualization tools amongst the research community and with developers. In this paper, we present our own experiences of "plugging-in" our visualization tool, SHriMP Views, into this environment. The Eclipse platform's Java Development Tools (JDT) and CVS plug-ins provide us with invaluable information on software artifacts relieving us from the burden of creating this functionality from scratch. This allows us to focus our efforts on the quality of our visualizations and, as our tool is now part of a full-featured Java IDE, gives us greater opportunities to evaluate our visualizations. The integration process required us to re-think some of our tool's architecture, strengthening its ability to be plugged into other environments. We step through a real-life scenario, using our newly integrated tool to aid us in merging of two branches of source code. Finally we detail some of the issues we have encountered in this integration and provide recommendations for other developers of visualization tools considering integration with the Eclipse platform. (30 refs)
Main heading: Computer aided software engineering
Controlled terms: Codes (symbols) - Graphical user interfaces - Java programming language

Uncontrolled terms: Software visualization
Classification Code: 722.2 Computer Peripheral Equipment - 723.1 Computer Programming - 723.1.1 Computer Programming Languages - 723.2 Data Processing and Image Processing - 723.5 Computer Applications
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

13. Building up and reasoning about architectural knowledge

Kruchten, Philippe (1); Lago, Patricia (2); Van Vliet, Hans (2)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 4214 LNCS, p 43-58, 2006, *Quality of Software Architectures - Second International Conference on Quality of Software Architectures, QoSA 2006, Revised Papers*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3540488197, **ISBN-13:** 9783540488194; **DOI:** 10.1007/11921998_8; **Conference:** 2nd International Conference on Quality of Software Architectures, QoSA 2006, June 27, 2006 - June 29, 2006; **Sponsor:** University of Karlsruhe; Malardalen University; Vasteras City; **Publisher:** Springer Verlag
Author affiliation: (1) University of British Columbia, Vancouver, BC, Canada (2) Vrije Universiteit, Amsterdam, Netherlands
Abstract: Architectural knowledge consists of architecture design as well as the design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is. Except for the architecture design part, most of the architectural knowledge usually remains hidden, tacit in the heads of the architects. We conjecture that an explicit representation of architectural knowledge is helpful for building and evolving quality systems. If we had a repository of architectural knowledge for a system, what would it ideally contain, how would we build it, and exploit it in practice? In this paper we describe a use-case model for an architectural knowledge base, together with its underlying ontology. We present a small case study in which we model available architectural knowledge in a commercial tool, the Aduna Cluster Map Viewer, which is aimed at ontology-based visualization. Putting together ontologies, use cases and tool support, we are able to reason about which types of architecting tasks can be supported, and how this can be done. © 2006 Springer-Verlag. (27 refs)
Main heading: Software architecture
Controlled terms: Computer software selection and evaluation - Knowledge based systems - Knowledge representation - Ontology
Uncontrolled terms: Architectural knowledge - Architectural knowledge base - Architecture designs - Commercial tools - Design decisions - Explicit representation - Ontology-based - Particular solution - Quality systems - Tool support - Use case model
Classification Code: 912.2 Management - 903 Information Science - 723.5 Computer Applications - 723.4.1 Expert Systems - 723.4 Artificial Intelligence - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

14. Quality attribute conflicts - Experiences from a large telecommunication application

Häggander, D. (1); Lundberg, L. (1); Matton, J. (1)
Source: *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, p 96-105, 2001; **DOI:** 10.1109/ICECCS.2001.930168; **Conference:** 7th IEEE International Conference on Engineering of Complex Computer Systems, June 11, 2001 - June 13, 2001; **Sponsor:** IEEE; University of Skovde; Municipality of Skovde; ARTES; **Publisher:** Institute of Electrical and Electronics Engineers Inc.
Author affiliation: (1) Department of Software Engineering, Blekinge Institute of Technology, Box 520, S-372 25 Ronneby, Sweden
Abstract: Modern telecommunication applications must provide high availability and performance. They must also be maintainable in order to reduce the maintenance cost and time-to-market for new versions. Previous studies have shown that the ambition to build maintainable systems may result in very poor performance. Here we evaluate an application called SDP pre-paid and show that the ambition to build systems with high performance and availability can lead to a complex software design with poor maintainability. We show that more than 85% of the SDP code is due to performance and availability optimizations. By implementing a SDP prototype with an alternative architecture we show that the code size can be reduced with an order of magnitude by removing the performance and availability optimizations from the source code and instead using modern fault tolerant hardware and third party software. The performance and availability of the prototype is as least as good as the old SDP. The hardware and third party software

cost is only 20-30% higher for the prototype. We also define three guidelines that help us to focus the additional hardware investments to the parts where it is really needed. (16 refs)
Main heading: Quality of service
Controlled terms: Computer hardware - Computer software maintenance - Optimization - Performance - Telecommunication systems
Uncontrolled terms: Quality attribute conflicts - Software design
Classification Code: 716.1 Information Theory and Signal Processing - 722 Computer Systems and Equipment - 723.5 Computer Applications - 921.5 Optimization Techniques
Treatment: Applications (APP)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

15. Comparing software architecture descriptions and raw source-code: A statistical analysis of maintainability metrics

Anjos, Eudisley (1, 2); Castor, Fernando (3); Zenha-Rela, Mário (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 7973 LNCS, n PART 3, p 199-213, 2013, *Computational Science and Its Applications, ICCSA 2013 - 13th International Conference, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-13:** 9783642396458; **DOI:** 10.1007/978-3-642-39646-5_15; **Conference:** 13th International Conference on Computational Science and Its Applications, ICCSA 2013, June 24, 2013 - June 27, 2013; **Sponsor:** Ho CHi Minh City International University; University of Perugia; Monash University; Kyushu Sangyo University; University of Basilicata; The Office of Naval Research; **Publisher:** Springer Verlag
Author affiliation: (1) CISUC, Centre for Informatics and Systems, University of Coimbra, Portugal (2) CI, Informatic Center, Federal University of Paraiba, Brazil (3) Cin, Informatic Center, Federal University of Pernambuco, Brazil
Abstract: The software systems have been exposed to constant changes in a short period of time. It requires high maintainable systems and makes maintainability one of the most important quality attributes. In this work we performed a statistical analysis of maintainability metrics in three mainstream open-source applications, Tomcat (webserver), Jedit (text editor) and Vuze (a peer to peer client). The metrics are applied to source-code and to derived similar architectural metrics using scatter plot, Pearson's correlation coefficient and significance tests. The observations contradict the common assumption that software quality attributes (aka non-functional requirements) are mostly determined at the architectural level and raise new issues for future works in this field. © 2013 Springer-Verlag Berlin Heidelberg. (33 refs)
Main heading: Peer to peer networks
Controlled terms: Computer software selection and evaluation - Correlation methods - Maintainability
Uncontrolled terms: Architectural levels - Architecture description - Non-functional requirements - Pearson's correlation coefficients - Quality attributes - Significance test - Software quality attributes - Software systems
Classification Code: 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 913.5 Maintenance - 922.2 Mathematical Statistics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

16. Improving the quality of software by quantifying the code change metric and predicting the bugs

Singh, V.B. (1); Chaturvedi, K.K. (2)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 7972 LNCS, n PART 2, p 408-426, 2013, *Computational Science and Its Applications, ICCSA 2013 - 13th International Conference, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-13:** 9783642396427; **DOI:** 10.1007/978-3-642-39643-4_30; **Conference:** 13th International Conference on Computational Science and Its Applications, ICCSA 2013, June 24, 2013 - June 27, 2013; **Sponsor:** Ho CHi Minh City International University; University of Perugia; Monash University; Kyushu Sangyo University; University of Basilicata; The Office of Naval Research; **Publisher:** Springer Verlag
Author affiliation: (1) Delhi College of Arts, Commerce University of Delhi, Delhi, India (2) Indian Agricultural Statistics Research Institute, New Delhi, India
Abstract: "When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to

the stage of science." LORD WILLIAM KELVIN (1824 - 1907). During the last decade, the quantification of software engineering process has got a pace due to availability of a huge amount of software repositories. These repositories include source code, bug, communication among developers/users, changes in code, etc. Researchers are trying to find out useful information from these repositories for improving the quality of software. The absence of bugs in the software is a major factor that decides the quality of software. In the available literature, researchers have proposed and implemented a plethora of bug prediction approaches varying in terms of accuracy, complexity and input data. The code change metric based bug prediction is proven to be very useful. In the literature, decay functions have been proposed that decay the complexity of code changes over a period of time in either exponential or linear fashion but they do not fit in open source software development paradigm because in open source software development paradigm, the development team is geographical dispersed and there is an irregular fluctuation in the code changes and bug detection/fixing process. The complexity of code changes reduces over a period of time that may be less than exponential or more than linear. This paper presents the method that quantifies the code change metric and also proposed decay functions that capture the variability in the decay curves represented the complexity of code changes. The proposed decay functions model the complexity of code changes which reduces over a period of time and follows different types of decay curves. We have collected the source code change data of Mozilla components and applied simple linear regression (SLR) and support vector regression (SVR) techniques to validate the proposed method and predict the bugs yet to come in future based on the current year complexity of code changes (entropy). The performance of proposed models has been compared using different performance criteria namely R2, Adjusted R2, Variation and Root Mean Squared Prediction Error (RMSPE). © 2013 Springer-Verlag Berlin Heidelberg. (34 refs)
Main heading: Program debugging
Controlled terms: Computer software selection and evaluation - Decay (organic) - Entropy - Forecasting - Software design
Uncontrolled terms: Bug predictions - Code changes - Open source software development - Simple linear regression - Software engineering process - Software Quality - Software repositories - Support vector regression (SVR)
Classification Code: 641.1 Thermodynamics - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 811.2 Wood and Wood Products - 921 Mathematics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

17. Knowledge based quality-driven architecture design and evaluation

Ovaska, Eila (1); Evesti, Antti (1); Henttonen, Katja (1); Palviainen, Marko (1); Aho, Pekka (1)
Source: *Information and Software Technology*, v 52, n 6, p 577-601, June 2010; **ISSN:** 09505849; **DOI:** 10.1016/j.infsof.2009.11.008; **Publisher:** Elsevier
Author affiliation: (1) VTT Technical Research Centre of Finland, Kaitoväylä 1, 90570 Oulu, Finland
Abstract: Modelling and evaluating quality properties of software is of high importance, especially when our every day life depends on the quality of services produced by systems and devices embedded into our surroundings. This paper contributes to the body of research in quality and model driven software engineering. It does so by introducing; (1) a quality aware software architecting approach and (2) a supporting tool chain. The novel approach with supporting tools enables the systematic development of high quality software by merging benefits of knowledge modelling and management, and model driven architecture design enhanced with domain-specific quality attributes. The whole design flow of software engineering is semi-automatic; specifying quality requirements, transforming quality requirements to architecture design, representing quality properties in architectural models, predicting quality fulfilment from architectural models, and finally, measuring quality aspects from implemented source code. The semi-automatic design flow is exemplified by the ongoing development of a secure middleware for peer-to-peer embedded systems. © 2009 Elsevier B.V. All rights reserved. (84 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Architecture - Design - Embedded software - Embedded systems - Knowledge based systems - Middleware - Models - Ontology - Peer to peer networks - Software architecture
Uncontrolled terms: Architectural models - Architecture design and evaluation - Architecture designs - Design flows - Domain specific - Evaluation - Evaluation models - High-quality software - Knowledge modelling - Model driven architectures - Model driven software engineering - Peer to peer - Quality aspects - Quality attribute - Quality attributes - Quality properties - Quality requirements - Semi-automatics - Software architecting - Source codes - Supporting tool
Classification Code: 903 Information Science - 902.1 Engineering Graphics - 723.5 Computer Applications - 723.4.1 Expert Systems - 912.2 Management - 723.1 Computer Programming - 722 Computer Systems and Equipment - 408 Structural Design - 402 Buildings and Towers - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

18. A Framework for automatically mining source code

Khatoon, Shaheen (1); Li, Guohui (1); Ashfaq, Rana Muhammad (2)
Source: *Journal of Software Engineering*, v 5, n 2, p 64-77, 2011; **ISSN:** 18194311; **DOI:** 10.3923/jse.2011.64.77;
Publisher: Academic Journals Inc.
Author affiliation: (1) School of Computer and Applied Technology, Huazhong University of Science and Technology (HUST), Wuhan, China (2) Department of Computer Science, International Islamic University, Islamabad, Pakistan
Abstract: Mining source code by using different data mining techniques to extract the informative patterns like programming rules, variable correlation, code clones and frequent API usage is an active area of research. However, no practical framework for integrating these tasks has been attempted. To achieve this objective an integrated framework is designed that can detect different types of bugs to achieve software quality and assist developer in reusing API libraries for rapid software development. Proposed framework automatically extracts large variety of programming patterns and finds the locations where the extracted patterns are violated. Violated patterns are reported as programming rule violation, copy paste code related bugs and inconsistent variable update bugs. Although, the bugs are different but the framework can detect these bugs in one pass and produces higher quality software systems within budget. The framework also helps in code reusing by suggesting the programmer how to write API code to facilitate rapid software development. Proposed framework is validated by developing a prototype that developed in C# (MS Visual Studio, 2008) and evaluated on large application like ERP. Results shows proposed technique greatly reduced time and cost of manually checking defects from source code by programmers. © 2011 Academic Journals Inc. (32 refs)
Main heading: Program debugging
Controlled terms: Application programming interfaces (API) - Codes (symbols) - Computer software selection and evaluation - Data mining - Java programming language - Software design
Uncontrolled terms: Active area - API usage - Code clone - Constraint-based mining - Copy-paste code - Data mining techniques - Higher quality softwares - Informative patterns - Integrated frameworks - One-pass - Programming patterns - Rule violation - Software Quality - Source codes - Source-code mining - Visual studios
Classification Code: 723 Computer Software, Data Handling and Applications
Database: Compendex

Data Provider: Engineering Village

19. Software quality in artificial intelligence system

Vinayagasundaram, B. (1); Srivatsa, S.K. (1)
Source: *Information Technology Journal*, v 6, n 6, p 835-842, August 15, 2007; **ISSN:** 18125638, **E-ISSN:** 18125646;
Publisher: Asian Network for Scientific Information
Author affiliation: (1) Computer Center, MIT Campus, Anna University, Chromepet Chennai-600044, Tamilnadu, India
Abstract: The main objective of the study is to define the metrics to measure the quality of software in the architecture for an artificial intelligence system. The proposed architecture for measurement consists of four components; Task specification layer, problem solver layer, domain layer and an adapter layer. These four components are hierarchically organized in a layered fashion. In this architecture, the overall structure is decomposed into sub components, in a layered way such that a new layer can be added to the existing layer that can change the behavior of the system.
The quality of components in the architecture are measured with metrics such as source code, depth of inheritance, number of paths, complexity level etc., These metrics are related to software quality characteristics suggested by ISO. This study is organized in the following way; Firstly, the study addresses the significance of software architecture in a software intensive AI system, the importance of quality of the software in the architecture and a layered architecture for artificial intelligence system. The secondly, the study addresses the relation ship between the quality characteristics and the metrics used for measuring the quality. The performance of the system with respect to functional requirement and nonfunctional requirements are measured and discussed. © 2007 Asian Network for Scientific Information. (11 refs)
Main heading: Software architecture
Controlled terms: Artificial intelligence - Quality control - Specifications
Uncontrolled terms: Nonfunctional requirements - Quality of software
Classification Code: 723.1 Computer Programming - 723.4 Artificial Intelligence - 723.5 Computer Applications - 902.2 Codes and Standards - 913.3 Quality Assurance and Control
Treatment: Theoretical (THR)
Database: Compendex

Data Provider: Engineering Village

20. Platform maintenance process for software quality assurance in product line

Jong, Sung Dong (1); Keun, Lee (1); Kyong, Hwan Kim (1); Sang, Tae Kim (1); Ji, Man Cho (1); Te, Hi Kim (1)
Source: *Proceedings - International Conference on Computer Science and Software Engineering, CSSE 2008*, v 2, p 325-331, 2008, *Proceedings - International Conference on Computer Science and Software Engineering, CSSE 2008*; **ISBN-13:** 9780769533360; **DOI:** 10.1109/CSSE.2008.1520; **Article number:** 4722063; **Conference:** International Conference on Computer Science and Software Engineering, CSSE 2008, December 12, 2008 - December 14, 2008; **Publisher:** IEEE Computer Society
Author affiliation: (1) Software Laboratory, CTO, Samsung Electronics
Abstract: As software products increase and continuously evolve, the improvement of software quality and increase of productivity are becoming crucial in consumer electronics. The software product line method, which develops software products by reusing software architecture and source code of product families rather than making separate software products, is gaining importance to achieve rapid responsiveness to market demand. As the software product line method becomes a core technology in consumer electronics, an issue that inevitably arises due to the reuse of software architecture is the management of platform quality through defect removal and continuous maintenance. This paper presents the details of the Platform Maintenance Process established to achieve such purpose, and validates the process through application in a business division which resulted in an approximate of 30% reduction in average defect fix time. © 2008 IEEE. (18 refs)
Main heading: Software architecture
Controlled terms: Computer science - Computer software maintenance - Computer software reusability - Computer software selection and evaluation - Consumer electronics - Defects - Production engineering - Quality assurance - Software design
Uncontrolled terms: Continuous maintenance - Core technology - Maintenance process - Market demand - Product families - Product-lines - Software maintenance - Software Product Line - Software product line engineering - Software products - Software Quality - Software quality assurance - Source codes
Classification Code: 423 Non Mechanical Properties and Tests of Building Materials - 715 Electronic Equipment, General Purpose and Industrial - 723 Computer Software, Data Handling and Applications - 913 Production Planning and Control; Manufacturing - 951 Materials Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

21. RepoGuard: A framework for integration of development tools with source code repositories

Legenhausen, Malte (1); Pielicke, Stefan (1); Rühmkorf, Jens (1); Wendel, Heinrich (1); Schreiber, Andreas (1)
Source: *Proceedings - 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009*, p 328-331, 2009, *Proceedings - 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009*; **ISBN-13:** 9780769537108; **DOI:** 10.1109/ICGSE.2009.51; **Article number:** 5196955; **Conference:** 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009, July 13, 2009 - July 16, 2009; **Sponsor:** Infosys; Lero; sfi; **Publisher:** IEEE Computer Society
Author affiliation: (1) Simulation and Software Technology, Deutsches Zentrum für Luft- und Raumfahrt E.v. (DLR), Linder Höhe, 51147 Cologne, Germany
Abstract: Today modern software development is not possible without the aid of tools like version control systems, bug tracking systems or instruments that ensure the compliance with code conventions. Unfortunately, all of these tools live in their own world", are only loosely coupled and do not interact with each other. RepoGuard addresses this problem by linking version control systems to other software development tools. It is implemented as an extension to several version control systems and provides interfaces to integrate other tools. The use of RepoGuard allows maximum control and validation of all committed resources before they are permanently stored. Additionally, RepoGuard provides communication channels in order to inform all relevant stakeholders about the failure or success of the process. Overall, RepoGuard provides simple but effective means to guarantee software quality standards in distributed development processes. © 2009 IEEE. (8 refs)
Main heading: Standardization
Controlled terms: Computer software selection and evaluation - Control systems - Software design
Uncontrolled terms: Bug tracking system - Code conventions - Communication channel - Development tools - Distributed development - Software development - Software development tools - Software Quality - Source code repositories - Version control system
Classification Code: 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.5 Computer Applications - 731.1 Control Systems - 902.2 Codes and Standards - 912.2 Management
Database: Compendex

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

22. Designing test engine for computer-aided software testing tools

Ma, Xue-Ying (1); Sheng, Bin-Kui (1)
Source: *WSEAS Transactions on Computers*, v 10, n 5, p 135-145, May 2011; **ISSN:** 11092750; **Publisher:** World Scientific and Engineering Academy and Society
Author affiliation: (1) College of Information Management, Zhejiang University of Finance and Economics, Hangzhou, China
Abstract: With the rapid development of software scale and programming languages, it is impossible to test software manually. The case for automating the software testing process has been made repeatedly and convincingly by numerous testing professionals. Automated tests can promote the efficiency of software testing and then to increase software productivity, improve software quality, and reduce cost in almost all processes of software engineering. White-box testing is one of the most important software testing strategies that can detect error even when the software specification is vague or incomplete. This paper gives a detailed description of the design and implementation of a testing engine. The testing engine, which is the kernel of a developed structured software-testing tool for the Visual Basic and C/C++ language, mainly consists of three components: program analyzer, source code instrumentation tool and intermediate database. In the testing engine, a block division mechanism and a new block-based CFG model are introduced and some block-based test adequacy criteria are extended. The programs are divided into a sequence of blocks and then instrumented and compiled in the testing engine, and all the information related to the test is saved in the intermediate database. The testing engine, acting as an agency, associates the testing automation module with instrumented executable program rather than the source code, and therefore the testing tool can easily be developed to accommodate new requirements and different testing adequacy criteria. It is also convenient to build a testing environment for multi-languages by modifying the program analyzer only, due to the flexibility of the software architecture. (16 refs)
Main heading: Software testing
Controlled terms: Computer programming languages - Computer software selection and evaluation - Database systems - Engines - Instruments - Software architecture - Software design
Uncontrolled terms: Automated test - Block division - Executable programs - Intermediate database - Object-oriented software-testing - Program instrumentation - Rapid development - Software productivity - Software Quality - Software Specification - Software test - Source Code Instrumentation - Source codes - Test adequacy criteria - Testing automation - Testing environment - Testing strategies - Testing tools - Three component - VISUAL BASIC - White-box testing
Classification Code: 612 Engines - 723 Computer Software, Data Handling and Applications - 941 Acoustical and Optical Measuring Instruments - 942 Electric and Electronic Measuring Instruments - 943 Mechanical and Miscellaneous Measuring Instruments - 944 Moisture, Pressure and Temperature, and Radiation Measuring Instruments
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

23. A distributed name resolution system to provide application layer identifier

Sawada, Atsushi (1); Noro, Masami (1); Hachisu, Yoshinari (1); Chang, Han-Myung (1); Yoshida, Atsushi (1); Osa, Daisuke (2); Urano, Akihiko (2)
Source: *Computer Software*, v 28, n 4, p 241-261, 2011; **Language:** Japanese; **ISSN:** 02896540; **Publisher:** Japan Society for Software Science and Technology
Author affiliation: (1) Department of Software Engineering, Nanzan University, Japan (2) Graduate School of Mathematical Sciences and Information Engineering, Nanzan University, Hitachi Solutions, Ltd., Japan
Abstract: In this paper, we discuss the design and evolution of the software architecture for source code inspection tools. Since there are a variety of demands on software quality improvement through source code inspection techniques, a tool for code inspection is required to be flexible enough to keep up with various needs of various users. We have developed JCI (Java Code Inspector): a source code inspection tool for Java, through a threeyear- long joint industry-university project which we call OJL (On the Job Learning). In this project, we have designed the software architecture of JCI using the GoF design patterns to realize analyzability, changeability, testability and efficiency. In this paper, we discuss the validity of our software architecture design as a foundation on which we can develop and evolve source code inspection tools, through several types of changes which have been carried out to deal with changing requirements of users. (13 refs)
Main heading: Software architecture

Controlled terms: Codes (symbols) - Computer programming languages - Computer software selection and evaluation - Design - Inspection - Inspection equipment - Machine tools - Software design
Uncontrolled terms: Application layers - Code inspections - Design Patterns - Java codes - Name resolution - Software architecture design - Software quality improvements - Source codes - Testability
Classification Code: 408 Structural Design - 603.1 Machine Tools, General - 723 Computer Software, Data Handling and Applications - 913.3.1 Inspection
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

24. A model to identify refactoring effort during maintenance by mining source code repositories

Moser, Raimund (1); Pedrycz, Witold (2); Sillitti, Alberto (1); Succi, Giancarlo (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 5089 LNCS, p 360-370, 2008, *Product-Focused Software Process Improvement - 9th International Conference, PROFES 2008, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3540695648, **ISBN-13:** 9783540695646; **DOI:** 10.1007/978-3-540-69566-0_29; **Conference:** 9th International Conference on Product-Focused Software Process Improvement, PROFES 2008, June 23, 2008 - June 25, 2008; **Publisher:** Springer Verlag
Author affiliation: (1) Center for Applied Software Engineering, Free University of Bolzano-Bozen, Italy (2) Department of Electrical and Computer Engineering, University of Alberta, Canada
Abstract: The use of refactoring as a way to continuously improve the design and quality of software and prevent its aging is mostly limited to Agile Methodologies and to a lower amount to software reengineering. In these communities refactoring is supposed to improve in the long-term the structure of existing code in order to make it easier to modify and maintain. To sustain such claims and analyze the impact of refactoring on maintenance we need to know how much refactoring developers do. In few cases such information is directly available for example from CVS log messages. In this study we propose a model on how to mine software repositories in order to obtain information of refactoring effort throughout the evolution of a software system. Moreover, we have developed a prototype that implements our model and validate our approach with two small case studies. © 2008 Springer-Verlag Berlin Heidelberg. (20 refs)
Main heading: Computer software maintenance
Controlled terms: Codes (standards) - Codes (symbols) - Computer software - Maintenance - Mining - Reengineering - Software design - Software engineering
Uncontrolled terms: Agile Methodologies - Case studies - International conferences - Know-how - Quality of softwares - Refactoring - Software evolution - Software metrics - Software Process Improvement - Software reengineering - Software repositories - Software systems - Source code repositories
Classification Code: 913.5 Maintenance - 913.3 Quality Assurance and Control - 902.2 Codes and Standards - 723.2 Data Processing and Image Processing - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 502.1 Mine and Quarry Operations
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

25. AODE for source code metrics for improved software maintainability

Yingjie, Tian (1); Chuanliang, Chen (2); Chunhua, Zhang (3)
Source: *Proceedings of the 4th International Conference on Semantics, Knowledge, and Grid, SKG 2008*, p 330-335, 2008, *Proceedings of the 4th International Conference on Semantics, Knowledge, and Grid, SKG 2008*; **ISBN-13:** 9780769534015; **DOI:** 10.1109/SKG.2008.43; **Article number:** 4725932; **Conference:** 4th International Conference on Semantics, Knowledge, and Grid, SKG 2008, December 3, 2008 - December 5, 2008; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) Research Centre on Fictitious Economy and Data Science, Chinese Academy of Sciences, Beijing 100080, China (2) Department of Computer Science, Beijing Normal University, Beijing 100875, China (3) School of Information, Renmin University of Chin, Beijing 100872, China
Abstract: Software metrics are collected at various phases of the whole software development process, in order to assist in monitoring and controlling the software quality. However, software quality control is complicated, because of the complex relationship between these metrics and the attributes of a software development process. To solve this problem, many excellent techniques have been introduced into software maintainability domain. In this paper, we propose a novel classification method-Aggregating One-Dependence Estimators (AODE) to support and enhance

our understanding of software metrics and their relationship to software quality. Experiments show that performance of AODE is much better than eight traditional classification methods and it is a promising method for software quality prediction. Furthermore, we present a Symmetrical Uncertainty (SU) based feature selection method to reduce source code metrics taking part in classification, make these classifiers more efficient and keep their performances not undermined meanwhile. Our empirical study shows the promising capability of SU for selecting relevant metrics and preserving original performances of the classifiers. © 2008 IEEE. (26 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Classifiers - Concurrency control - Feature extraction - Information theory - Learning systems - Maintainability - Quality assurance - Quality control - Quality function deployment - Semantics - Software design - Total quality management
Uncontrolled terms: Classification , - Classification methods - Complex relationships - Empirical studies - Feature selection methods - Monitoring and controlling - Software development process - Software maintainabilities - Software metrics - Software qualities - Software quality controls - Software quality predictions - Source codes
Classification Code: 922.2 Mathematical Statistics - 731.5 Robotics - 741.1 Light/Optics - 751.1 Acoustic Waves - 802.1 Chemical Plants and Equipment - 903.2 Information Dissemination - 912.2 Management - 913.3 Quality Assurance and Control - 913.5 Maintenance - 731.3 Specific Variables Control - 461.4 Ergonomics and Human Factors Engineering - 716 Telecommunication; Radar, Radio and Television - 716.1 Information Theory and Signal Processing - 723.5 Computer Applications - 723 Computer Software, Data Handling and Applications - 723.3 Database Systems - 723.4 Artificial Intelligence - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

26. E-quality: A graph based object oriented software quality visualization tool

Erdemir, Ural (1); Tekin, Umut (1); Buzluca, Feza (2)
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.2011.6069454; **Article number:** 6069454; **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) Center of Research for Advanced Technologies of Informatics and Information Security, Kocaeli, Turkey (2) Computer Engineering Department, Istanbul Technical University, Istanbul, Turkey
Abstract: Recently, with increasing maintenance costs, studies on software quality are becoming increasingly important and widespread because high quality software means more easily maintainable software. Measurement plays a key role in quality improvement activities and metrics are the quantitative measurement of software design quality. In this paper, we introduce a graph based object-oriented software quality visualization tool called "E- Quality". E-Quality automatically extracts quality metrics and class relations from Java source code and visualizes them on a graph-based interactive visual environment. This visual environment effectively simplifies comprehension and refactoring of complex software systems. Our approach assists developers in understanding of software quality attributes by level categorization and intuitive visualization techniques. Experimental results show that the tool can be used to detect software design flaws and refactoring opportunities. © 2011 IEEE. (30 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Computer software maintenance - Java programming language - Quality control - Software design - Visualization
Uncontrolled terms: Design Patterns - Eclipse - Flaw detection - Object-Oriented Metrics - Refactorings - Software Quality - Software Visualization
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics - 913.3 Quality Assurance and Control
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

27. Artificial intelligence for software quality improvement

Agüero, Martín (1); Madou, Franco (1); Esperón, Gabriela (2); de Luise, Daniela López (1)
Source: *World Academy of Science, Engineering and Technology*, v 63, p 234-239, March 2010; **ISSN:** 2010376X, **E-ISSN:** 20103778; **Publisher:** WASET - World Academy of Science, Engineering and

Author affiliation: (1) Universidad de Palermo, Argentina (2) Department of Exact Sciences, Engineering School, Universidad de Palermo, Argentina
Abstract: This paper presents a software quality support tool, a Java source code evaluator and a code profiler based on computational intelligence techniques. It is Java prototype software developed by AI Group [1] from the Research Laboratories at Universidad de Palermo: an Intelligent Java Analyzer (in Spanish: Analizador Java Inteligente, AJI). It represents a new approach to evaluate and identify inaccurate source code usage and transitively, the software product itself. The aim of this project is to provide the software development industry with a new tool to increase software quality by extending the value of source code metrics through computational intelligence. (22 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Clustering algorithms - Expert systems - Neural networks - Problem solving - Research laboratories - Software design - Software prototyping
Uncontrolled terms: Computational intelligence - Computational intelligence techniques - Java source codes - New approaches - New tools - Prototype software - Software development - Software metrics - Software products - Software Quality - Software quality improvements - Source codes
Classification Code: 912.2 Management - 903.1 Information Sources and Analysis - 901.3 Engineering Research - 723.5 Computer Applications - 723.4.1 Expert Systems - 921 Mathematics - 723.4 Artificial Intelligence - 723 Computer Software, Data Handling and Applications - 721 Computer Circuits and Logic Elements - 461.1 Biomedical Engineering - 402 Buildings and Towers - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

28. Attractiveness of open source projects: A path to software quality

Santos, Carlos (1); Nelson, Kay (1)
Source: *Association for Information Systems - 13th Americas Conference on Information Systems, AMCIS 2007: Reaching New Heights*, v 6, p 3939-3953, 2007, *Association for Information Systems - 13th Americas Conference on Information Systems, AMCIS 2007: Reaching New Heights*; **ISBN-13:** 9781604233810; **Conference:** 13th Americas Conference on Information Systems, AMCIS 2007, August 10, 2007 - August 12, 2007; **Publisher:** AIS/ICIS Administrative Office
Author affiliation: (1) Southern Illinois University, United States
Abstract: The Open Source Software movement is impacting society and organizations in significant ways. This impact can be observed not only economically but also on the way business processes like software development are performed within organizations. The success of open source software is attributed to its practices and organizational structure. Consequently, there is a trend in the corporate environment to copy and adapt some of these practices, such as releasing software source code to the community, giving up proprietary rights. This paper is an attempt to model what are the drivers for success of this specific practice-trend of releasing source code to the community by corporations. Thus, to understand (1) what the specific sponsor's motivations to engage in projects intended to release source code are, thereby defining success of the practice, and (2) the conditions under which those projects are more likely to generate desirable outcomes become important to both researchers and practitioners. This paper pursues both of these topics, presenting propositions to empirically explain the numbers of contributions and contributors - attractiveness - a corporate project has observed and why. We assume that attractiveness leads to software quality, a condition desired by profit-oriented managers. (30 refs)
Main heading: Open systems
Controlled terms: Computer programming languages - Computer software selection and evaluation - Information dissemination - Information systems - Profitability - Software design - Software engineering
Uncontrolled terms: Business Process - Corporate environment - Interdependence - Open source projects - Open Source Software - Open sources - Organizational practices - Organizational structures - Proprietary rights - Software modules - Software Quality - Software source codes - Source codes
Classification Code: 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 903.2 Information Dissemination - 911.2 Industrial Economics
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

29. Visualization of software architecture graphs of Java systems: Managing propagated low level dependencies

Schrettner, Lajos (1); Fülöp, Lajos Jeno (1); Ferenc, Rudolf (1); Gyimóthy, Tibor (1)

Source: *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, PPPJ 2010*, p 148-157, 2010, *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, PPPJ 2010*; **ISBN-13:** 9781450302692; **DOI:** 10.1145/1852761.1852783; **Conference:** 8th Annual Conference on Principles and Practice of Programming in Java, PPPJ 2010, September 15, 2010 - September 17, 2010; **Sponsor:** ORACLE; Vienna University of Technology; computer languages; **Publisher:** Association for Computing Machinery
Author affiliation: (1) Department of Software Engineering, University of Szeged, Hungary
Abstract: The availability of up-to-date documentation of the architecture is crucial for software maintenance tasks, but it is often missing or differs from the implemented architecture. An increasingly popular and feasible way to get a clear picture of the architecture is to reconstruct it from the source code. The result of the reconstruction procedure is a graph with special, architecture-specific properties. Nowadays software systems are typically very large, so the reconstructed architecture contains a lot of details and is really difficult to interpret. It is important therefore to have efficient methods that help in understanding and managing the architecture graph. The purpose of these methods is to try to present the information so that it is comprehensible to the users. Two important methods are selective subtree collapsion and lifting low level dependencies of the system into higher, visible levels. These enable an architect to investigate the dependencies of system components at higher levels, without the need to deal with an enormous quantity of low-level details. In this paper, first we overview the concepts related to lifting and present a conceptual framework that combines subtree collapsion with lifting to enable users to interactively explore and manipulate a software architecture graph. Then we define a set of algorithms that can be used to efficiently propagate dependency edges of a graph to higher levels. We also describe how the results can be integrated into SourceInventory, a software quality monitoring and visualization framework. © 2010 ACM. (15 refs)
Main heading: Software architecture
Controlled terms: Computer software maintenance - Computer software selection and evaluation - Java programming language - Visualization
Uncontrolled terms: Architecture reconstruction - architecture visualization - Conceptual frameworks - Efficient method - Java system - lifting - Low level - nested set model - Reconstruction procedure - Software Quality - Software systems - Software-maintenance tasks - Source codes - Subtrees - System components - Visualization framework
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

30. Don't touch my code! Examining the effects of ownership on software quality

Bird, Christian (1); Nagappan, Nachiappan (1); Murphy, Brendan (1); Gall, Harald (2); Devanbu, Premkumar (3)
Source: *SIGSOFT/FSE 2011 - Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*, p 4-14, 2011, *SIGSOFT/FSE'11 - Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*; **ISBN-13:** 9781450304436; **DOI:** 10.1145/2025113.2025119; **Conference:** 19th ACM SIGSOFT Symposium on Foundations of Software Engineering, SIGSOFT/FSE'11, September 5, 2011 - September 9, 2011; **Sponsor:** Assoc. Comput. Mach., Spec.; Interest Group Softw. Eng. (ACM SIGSOFT); **Publisher:** Association for Computing Machinery
Author affiliation: (1) Microsoft Research, United States (2) University of Zurich, Switzerland (3) University of California, Davis, United States
Abstract: Ownership is a key aspect of large-scale software development. We examine the relationship between different ownership measures and software failures in two large software projects: Windows Vista and Windows 7. We find that in all cases, measures of ownership such as the number of low-expertise developers, and the proportion of ownership for the top owner have a relationship with both pre-release faults and post-release failures. We also empirically identify reasons that low-expertise developers make changes to components and show that the removal of low-expertise contributions dramatically decreases the performance of contribution based defect prediction. Finally we provide recommendations for source code change policies and utilization of resources such as code inspections based on our results. © 2011 ACM. (35 refs)
Main heading: Software design
Controlled terms: Computer software selection and evaluation - Image quality
Uncontrolled terms: Code inspections - Defect prediction - Empirical Software Engineering - Expertise - Ownership - Software failure - Software project - Software Quality - Source code changes - Windows Vista
Classification Code: 723 Computer Software, Data Handling and Applications - 741 Light, Optics and Optical Devices
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

31. Practice patterns to improve the quality of design model in embedded software development

Kim, Doo-Hwan (1); Kim, Jong-Phil (1); Hong, Jang-Eui (1)
Source: *Proceedings - International Conference on Quality Software*, p 179-184, 2009, *QSIC 2009 - Proceedings of the 9th International Conference on Quality Software*; **ISSN:** 15506002; **ISBN-13:** 9780769538280; **DOI:** 10.1109/QSIC.2009.32; **Article number:** 5381471; **Conference:** 9th International Conference on Quality Software, QSIC 2009, August 24, 2009 - August 25, 2009; **Sponsor:** Software Engineering Society of Korean; Institute for Information Scientists and Engineers; IEEE Reliability Society; KAIST (Korea Advanced Institute of Science and Technology); Korea Information Promotion Agency; Samsung SDS; **Publisher:** IEEE Computer Society
Author affiliation: (1) Software Engineering Laboratory, Chungbuk National University, Korea, Republic of
Abstract: Source code quality is very important to embedded systems because software embedded into a product is difficult to change. In order to improve source code quality, the quality of analysis and design models as well as the quality of source code should be considered because code quality can be improved by design model quality. With the reason, we suggest, in this paper, "Practice Pattern" as one of practical techniques to improve embedded software quality, which focus on improving the quality of software design model. Practice pattern is a process pattern to guide modeling activities in software development process. We believe that adopting our pattern provides the benefits of performance, modularization, and easy to implement for embedded software. © 2009 IEEE. (14 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Design - Embedded software - Embedded systems - Modular construction - Software design
Uncontrolled terms: Analysis and design models - Code quality - Design models - Embedded software development - Modularizations - Practice pattern - Process patterns - Quality of design - Quality of softwares - Software development process - Software Quality - Source codes
Classification Code: 902.1 Engineering Graphics - 723.5 Computer Applications - 723.1 Computer Programming - 912.2 Management - 723 Computer Software, Data Handling and Applications - 408 Structural Design - 405.2 Construction Methods - 722 Computer Systems and Equipment
Database: Compendex

32. Towards the open source reference architectures

Nakagawa, Elisa Yumi (1); Maldonado, José Carlos (1)
Source: *Proceedings - 5th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2011*, p 61-70, 2011, *Proceedings - 5th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2011*; **ISBN-13:** 9780769546261; **DOI:** 10.1109/SBCARS.2011.12; **Article number:** 6114567; **Conference:** 5th Brazilian Symposium on Components, Architectures and Reuse, SBCARS 2011, September 26, 2011 - September 27, 2011; **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computer Systems, University of São Paulo - USP, PO Box 668, 13560-970 São Carlos, SP, Brazil
Abstract: Software architectures have received increasing attention by playing a significant role in determining the success and quality of software systems. In particular, reference architecture is a special type of architecture that captures the essence of software systems of a specific domain, achieving therefore well-recognized understanding of that domain. In spite of this, it is not observed concern in order to widely disseminate reference architectures and, as a consequence, the knowledge encompassed by these architectures. At the same time, Open Source Software (OSS) has been largely developed and used in both academy and industry. A diversity of OSS has been made available and has contributed to the software development through dissemination of knowledge that is encompassed mainly in the source code. Specifically, its success is due to OSS licences that have adequately supported its development and evolution. Thus, applying this same idea in order to disseminate reference architectures seems to be very interesting. The main contribution of this paper is to propose Open Source Reference Architecture (OSRA) that, based on principles of OSS, aims at promoting dissemination and evolution of reference architectures, intending to contribute to a more effective software development. © 2011 IEEE. (58 refs)
Main heading: Open systems
Controlled terms: Architecture - Computer software - Software architecture - Software design
Uncontrolled terms: Open source license - Open Source Software - Open sources - Quality of softwares - Reference architecture - Software systems - Source codes
Classification Code: 402 Buildings and Towers - 723 Computer Software, Data Handling and Applications
Database: Compendex

33. DSL-based support for semi-automated architectural component model abstraction throughout the software lifecycle

Haitzer, Thomas (1); Zdun, Uwe (1)
Source: *QoSA'12 - Proceedings of the 8th International ACM SIGSOFT Conference on the Quality of Software Architectures*, p 61-70, 2012, *QoSA'12 - Proceedings of the 8th International ACM SIGSOFT Conference on the Quality of Software Architectures*; **ISBN-13:** 9781450313469; **DOI:** 10.1145/2304696.2304709; **Conference:** 8th International ACM SIGSOFT Conference on the Quality of Software Architectures, QoSA'12, June 25, 2012 - June 28, 2012; **Sponsor:** Special Interest Group on Software Engineering (ACM SIGSOFT); **Publisher:** Association for Computing Machinery
Author affiliation: (1) Software Architecture Group, Faculty of Computer Science, University of Vienna, Austria
Abstract: In this paper we present an approach for supporting the semi-automated abstraction of architectural models throughout the software lifecycle. It addresses the problem that the design and the implementation of a software system often drift apart as software systems evolve, leading to architectural knowledge evaporation. Our approach provides concepts and tool support for the semi-automatic abstraction of architectural knowledge from implemented systems and keeping the abstracted architectural knowledge up-to-date. In particular, we propose architecture abstraction concepts that are supported through a domainspecific language (DSL). Our main focus is on providing architectural abstraction specifications in the DSL that only need to be changed, if the architecture changes, but can tolerate non-architectural changes in the underlying source code. The DSL and its tools support abstracting the source code into UML component models for describing the architecture. Once the software architect has defined an architectural abstraction in the DSL, we can automatically generate UML component models from the source code and check whether the architectural design constraints are fulfilled by the models. Our approach supports full traceability between source code elements and architectural abstractions, and allows software architects to compare different versions of the generated UML component model with each other. We evaluate our research results by studying the evolution of architectural abstractions in different consecutive versions and the execution times for five existing open source systems. Copyright © 2012 ACM. (41 refs)
Main heading: Open systems
Controlled terms: Abstracting - Architectural design - Automation - Computer programming languages - Computer software - DSL - Life cycle - Software architecture
Uncontrolled terms: Architectural abstraction - Architectural components - Model transformation - Software Evolution - UML
Classification Code: 402 Buildings and Towers - 723 Computer Software, Data Handling and Applications - 731 Automatic Control Principles and Applications - 732 Control Devices - 903.1 Information Sources and Analysis - 913.1 Production Engineering
Database: Compendex

34. Case study on incremental software development

Liu, Dapeng (1); Xu, Shaochun (2); Du, Wencai (3)
Source: *Proceedings - 2011 9th International Conference on Software Engineering Research, Management and Applications, SERA 2011*, p 227-234, 2011, *Proceedings - 2011 9th International Conference on Software Engineering Research, Management and Applications, SERA 2011*; **ISBN-13:** 9780769544908; **DOI:** 10.1109/SERA.2011.43; **Article number:** 6065596; **Conference:** 9th ACIS International Conference on Software Engineering Research, Management and Applications, SERA 2011, August 10, 2011 - August 12, 2011; **Sponsor:** Int. Assoc. Comput. Inf. Sci. (ACIS); **Publisher:** IEEE Computer Society
Author affiliation: (1) Brain Technologies, Marine Del Rey, CA, United States (2) Department of Computer Science, Algoma University, Sault Ste Marie, ON, Canada (3) College of Information Science and Technology, Hainan University, Haikou, China
Abstract: It has been long recognized that software development in industry is often iterative while new requirements and challenges arise along with software evolution. One of general rules for software design is to degrade coupling between modules so that software architecture is flexible and scalable. Along with software evolution it is better to keep user interface stay similar which helps users to upgrade, and to keep programming interface consistent for the same functionality which is undoubtedly beneficial to programmers. In this paper, we discuss about these situations along a real development case study and propose a few systematic solutions which facilitate source code reuse and assure software quality. Detailed analysis and comparison are provided to show their effectiveness. © 2011 IEEE. (19 refs)
Main heading: Software design
Controlled terms: Computer programming - Computer software reusability - Computer software selection and evaluation - Engineering research - Research and development management - Software architecture - User interfaces

Uncontrolled terms: design patterns. - incremental software development - programming interface - Software evolution - Software Quality - Source codes
Classification Code: 722.2 Computer Peripheral Equipment - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 901.3 Engineering Research
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

35. Analyzing the tracing of requirements and source code during software development: A research preview

Delater, Alexander (1); Paech, Barbara (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 7830 LNCS, p 308-314, 2013, *Requirements Engineering: Foundation for Software Quality - 19th International Working Conference, REFSQ 2013, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-13:** 9783642374210; **DOI:** 10.1007/978-3-642-37422-7_22; **Conference:** 19th International Working Conference on Requirements Engineering: Foundation for Software Quality, REFSQ 2013, April 8, 2013 - April 11, 2013; **Sponsor:** PALUNO - The Ruhr Institute for Software Technology; BOSCH - Invented for Life; International Requirements Engineering Board (IREB); SOPHIST; itemis; **Publisher:** Springer Verlag
Author affiliation: (1) Institute of Computer Science, University of Heidelberg, Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
Abstract: [Context and motivation] Traceability links between requirements and code are often created after development, which can, for example, lead to higher development effort. To address this weakness, we developed in previous work an approach that captures traceability links between requirements and code as the development progresses by using artifacts from project management called work items. [Question/problem] It is important to investigate empirically what is the best way to capture such links and how these links are used during development. [Principal ideas/results] In order to link requirements, work items and code during development, we extended our approach from previous work by defining three traceability link creation processes. We are applying these processes in practice in a software development project conducted with undergraduate students. The results indicate that our approach creates correct traceability links between requirements and code with high precision/recall during development, while developers mainly used the third process to link work items after implementation. Furthermore, the students used a subset of the created traceability links for navigating between requirements and code during the early phase of the development project. [Contribution] In this paper, we report on preliminary empirical results from applying our approach in practice. © 2013 Springer-Verlag. (10 refs)
Main heading: Software design
Controlled terms: Computer software selection and evaluation - Project management - Requirements engineering - Software engineering - Students
Uncontrolled terms: code - Development project - requirement - Software development projects - trace - Traceability links - Undergraduate students - Work items
Classification Code: 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 912.2 Management - 912.4 Personnel
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

36. Securing opensource code via static analysis

Kannavara, Raghudeep (1)
Source: *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, p 429-436, 2012, *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*; **ISBN-13:** 9780769546704; **DOI:** 10.1109/ICST.2012.123; **Article number:** 6200135; **Conference:** 5th IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, April 17, 2012 - April 21, 2012; **Publisher:** IEEE Computer Society
Author affiliation: (1) Security Center of Excellence, Intel Corporation, United States
Abstract: Static code analysis (SCA) is the analysis of computer programs that is performed without actually executing the programs, usually by using an automated tool. SCA has become an integral part of the software development life cycle and one of the first steps to detect and eliminate programming errors early in the software development stage. Although SCA tools are routinely used in proprietary software development environment to ensure software quality, application of such tools to the vast expanse of open source code presents a forbidding albeit interesting challenge, especially when open source code finds its way into commercial software. Although there have been

recent efforts in this direction, in this paper, we address this challenge to some extent by applying static analysis on a popular open source project, i.e., Linux kernel, discuss the results of our analysis and based on our analysis, we propose an alternate workflow that can be adopted while incorporating open source software in a commercial software development process. Further, we discuss the benefits and the challenges faced while adopting the proposed alternate workflow. © 2012 IEEE. (19 refs)
Main heading: Static analysis
Controlled terms: Computer software selection and evaluation - Life cycle - Open systems - Software design - Software testing
Uncontrolled terms: Automated tools - Commercial software - Commercial software development - Integral part - Linux kernel - Open source projects - Open Source Software - Open-source - Open-source code - Programming errors - Proprietary software - Software development life cycle - Software Quality - Static code analysis
Classification Code: 722 Computer Systems and Equipment - 723 Computer Software, Data Handling and Applications - 723.5 Computer Applications - 913.1 Production Engineering
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

37. Sociotechnical coordination and collaboration in open source software

Bird, Christian (1)
Source: *IEEE International Conference on Software Maintenance, ICSM*, p 568-573, 2011, *Proceedings of the 27th IEEE International Conference on Software Maintenance, ICSM 2011*; **ISBN-13:** 9781457706646; **DOI:** 10.1109/ICSM.2011.6080832; **Article number:** 6080832; **Conference:** 27th IEEE International Conference on Software Maintenance, ICSM 2011, September 25, 2011 - September 30, 2011; **Publisher:** IEEE Computer Society
Author affiliation: (1) Microsoft Research, Redmond, WA, United States
Abstract: In the mid 90s, a new style of software development, termed open source software (OSS) has emerged and has originated large, mature, stable, and widely used software projects. As software continues to grow in size and complexity, so do development teams. Consequently, coordination and communication within these teams play larger roles in productivity and software quality. My dissertation focuses on the relationships between developers in large open source projects and how software affects and is affected by these relationships. Fortunately, source code repository histories, mailing list archives, and bug databases from OSS projects contain latent data from which we can reconstruct a rich view of a project over time and analyze these sociotechnical relationships. We present methods of obtaining and analyzing this data as well as the results of empirical studies whose goal is to answer questions that can help stakeholders understand and make decisions about their own teams. We answer questions such as "Do large OSS project really have a disorganized bazaar-like structure?" "What is the relationship between social and development behavior in OSS?" "How does one progress from a project newcomer to a full-fledged, core developer?" and others in an attempt to understand how large, successful OSS projects work and also to contrast them with projects in commercial settings. © 2011 IEEE. (35 refs)
Main heading: Open systems
Controlled terms: Computer software maintenance - Computer software selection and evaluation - Software design
Uncontrolled terms: Commercial settings - Development teams - Empirical studies - Mailing lists - Open source projects - Open Source Software - Socio-technical relationships - Sociotechnical - Software project - Software Quality - Source code repositories
Classification Code: 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

38. A probabilstic mathematical model to measure software regularity

Ghazarian, Arbi (1)
Source: *Proceedings of the IASTED International Conference on Software Engineering and Applications, SEA 2011*, p 192-199, 2011, *Proceedings of the IASTED International Conference on Software Engineering and Applications, SEA 2011*; **ISBN-13:** 9780889869066; **DOI:** 10.2316/P.2011.758-006; **Conference:** 15th IASTED International Conference on Software Engineering and Applications, SEA 2011, December 14, 2011 - December 16, 2011; **Sponsor:** Int. Assoc. Sci. Technol. Dev. (IASTED); **Publisher:** Acta Press
Author affiliation: (1) Department or Engineering, Division of Computing Studies, College of Technology and Innovation, Mesa, AZ 85212, United States
Abstract: Software developers, either in a planned or ad hod fashion, introduce regularities such as architectural, design or programming conventions, idioms, and patterns into software systems while developing source code. There

are numerous unverfied claims in the software engineering literature about the impacts of such regularities on software quality. To investigate how the degree of conformance to regularities in a software component affects its various quality attributes necessarily requires a measure of software regularity. Unfortunately, an extensive search failed to find a measure of regularity, or design consistency, in the software engineering literature. In this paper, we propose a model to measure a component's Degree of Regularity (DR), as well as its Regularity Density (RD). We evaluate the validity of the proposed metrics in a case study of regularity and defect-based quality attributes on three major components of an industrial large-scale software system. Results from this empirical study suggest that a component's defect-based measures tend to decrease as its regularity-based measures increase. This finding has the potential implication of increasing the reliability of software systems by minimizing the defect rates of software components through controlling their regularity. (11 refs)
Main heading: Software engineering
Controlled terms: Computer software selection and evaluation - Defect density - Defects - Mathematical models - Quality control
Uncontrolled terms: Defect rate - Degree of regularity - Design consistency - Empirical studies - Large-scale software systems - Quality attributes - Software component - Software developer - Software Quality - Software systems - Source codes - Source-code regularity
Classification Code: 951 Materials Science - 933.1 Crystalline Solids - 921 Mathematics - 913.3 Quality Assurance and Control - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 423 Non Mechanical Properties and Tests of Building Materials
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

39. Testing and Formal Verification of Service Oriented Architectures

Sloan, John C. (1); Khoshgoftaar, Taghi M. (1)
Source: *International Journal of Reliability, Quality and Safety Engineering*, v 16, n 2, p 137-162, April 2009; **ISSN:** 02185393; **DOI:** 10.1142/S0218539309003332; **Publisher:** World Scientific Publishing Co. Pte. Ltd
Author affiliation: (1) Department of Computer Science and Engineering, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, United States
Abstract: We examine two open engineering problems in the area of testing and formal verification of internet-enabled service oriented architectures (SOA). The first involves deciding when to formally and exhaustively verify versus when to informally and non-exhaustively test. The second concerns scalability limitations associated with formal verification, to which we propose a semi-formal technique that uses software agents. Finally, we assess how these findings can improve current software quality assurance practices. Addressing the first problem, we present and explain two classes of tradeoffs. External tradeoffs between assurance, performance, and flexibility are determined by the business needs of each application, whether it be in engineering, commerce, or entertainment. Internal tradeoffs between assurance, scale, and level of detail involve the technical challenges of feasibly verifying or testing an SOA. To help decide whether to exhaustively verify or non-exhaustively test, we present and explain these two classes of tradeoffs. Identifying a middle ground between testing and verification, we propose using software agents to simulate services in a composition. Technologically, this approach has the advantage of assuring the quality of compositions that are too large to exhaustively verify. Operationally, it supports testing these compositions in the laboratory without access to source code or use of network resources of third-party services. We identify and exploit the structural similarities between agents and services, examining how doing so can assure the quality of service compositions. © 2009 World Scientific Publishing Company. (42 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Architectural design - Chemical analysis - Information services - Model checking - Quality assurance - Quality control - Quality of service - Service oriented architecture (SOA) - Software agents - Total quality management - Web services
Uncontrolled terms: Agent-based testing - Assurance tradeoffs - Business needs - Engineering problems - Formal techniques - Formal verifications - Level of detail - Network resource - Semi-formal verification - Software quality assurance - Source codes - Structural similarity - Technical challenges
Classification Code: 723.5 Computer Applications - 801 Chemistry - 804 Chemical Products Generally - 723.1 Computer Programming - 903.4 Information Services - 913.3 Quality Assurance and Control - 922.2 Mathematical Statistics - 912.2 Management - 723 Computer Software, Data Handling and Applications - 408.1 Structural Design, General - 716 Telecommunication; Radar, Radio and Television - 717 Optical Communication - 402 Buildings and Towers - 718 Telephone Systems and Related Technologies; Line Communications - 721.2 Logic Elements - 722.4 Digital Computers and Systems - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory
Database: Compendex

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

40. Heuristics for discovering architectural violations

Maffort, Cristiano (1); Valente, Marco Tulio (1); Bigonha, Mariza (1); Anquetil, Nicolas (2); Hora, Andre (2)
Source: *Proceedings - Working Conference on Reverse Engineering, WCRE*, p 222-231, 2013, *Proceedings - 20th Working Conference on Reverse Engineering, WCRE 2013*; **ISSN:** 10951350; **ISBN-13:** 9781479929313; **DOI:** 10.1109/WCRE.2013.6671297; **Article number:** 06671297; **Conference:** 20th Working Conference on Reverse Engineering, WCRE 2013, October 14, 2013 - October 17, 2013; **Sponsor:** The Reengineering Forum; Technical Council on Software Engineering (TCSE); **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computer Science, UFMG, Brazil (2) RMoD Project-Team, INRIA, Lille Nord Europe, France
Abstract: Software architecture conformance is a key software quality control activity that aims to reveal the progressive gap normally observed between concrete and planned software architectures. In this paper, we present ArchLint, a lightweight approach for architecture conformance based on a combination of static and historical source code analysis. For this purpose, ArchLint relies on four heuristics for detecting both absences and divergences in source code based architectures. We applied ArchLint in an industrial-strength system and as a result we detected 119 architectural violations, with an overall precision of 46.7% and a recall of 96.2%, for divergences. We also evaluated ArchLint with four open-source systems, used in an independent study on reflexion models. In this second study, ArchLint achieved precision results ranging from 57.1% to 89.4%. © 2013 IEEE. (28 refs)
Main heading: Software architecture
Controlled terms: Architecture - Computer software selection and evaluation - Reverse engineering - Static analysis
Uncontrolled terms: Mining software repositories - Open source system - Software quality control - Source code analysis - Source codes
Classification Code: 402 Buildings and Towers - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

41. Measuring OO design metrics from UML

Tang, Mei-Huei (1); Chen, Mei-Hwa (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 2460 LNCS, p 368-382, 2002, *UML 2002 - The Unified Modeling Language: Model Engineering, Concepts, and Tools - 5th International Conference, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3540442545, **ISBN-13:** 9783540442547; **Conference:** 5th International Conference on Unified Modeling Language: Model Engineering, Concepts, and Tools, UML 2002, September 30, 2002 - October 4, 2002; **Sponsor:** ACM Special Interest Group on Software Engineering (SIGSOFT); IEEE Computer Society; **Publisher:** Springer Verlag
Author affiliation: (1) Computer Science Department, SUNY at Albany, Albany, NY 12222, United States
Abstract: Design metrics are useful means for improving the quality of software. A number of object-oriented metrics have been suggested as being helpful for resource allocation in software development. These metrics are particularly useful for identifying fault-prone classes and for predicting required maintenance efforts, productivity, and rework efforts. To obtain the design metrics of the software under development, most existing approaches measure the metrics by parsing the source code of the software. Such approaches can only be performed in a late phase of software development, thus limiting the usefulness of the design metrics in resource allocation. In this paper, we present a methodology that compiles UML specifications to obtain design information and to compute the design metrics at an early stage of software development. The current version of our tool uses diagrams produced by the Rational Rose tool and computes OO metrics that have been suggested as being good indicators for identifying faults related to Object-Oriented features. Our technique advances the state of the metrics measuring process; thus it is expected to strongly promote the use of design metrics and significantly increase their impact on improving software quality. © Springer-Verlag Berlin Heidelberg 2002. (31 refs)
Main heading: Software design
Controlled terms: Computer software maintenance - Computer software selection and evaluation - Design - Resource allocation - Unified Modeling Language
Uncontrolled terms: Design information - Design metrics - Fault-prone - Object oriented metrics - Object-oriented features - OO metrics - Quality of softwares - Software development - Software Quality - Source codes - Tool use - UML specifications

Classification Code: 408 Structural Design - 723 Computer Software, Data Handling and Applications - 912.3 Operations Research
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

42. Bringing auto dynamic difficulty to commercial games: A reusable design pattern based approach

Chowdhury, Muhammad Iftekher (1); Katchabaw, Michael (1)
Source: *Proceedings of CGAMES 2013 USA - 18th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games*, p 103-110, 2013, *Proceedings of CGAMES 2013 USA - 18th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games*; **ISBN-13:** 9781479908189; **DOI:** 10.1109/CGames.2013.6632615; **Article number:** 6632615; **Conference:** 18th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games, CGAMES 2013, July 30, 2013 - August 1, 2013; **Sponsor:** IEEE Computer Society (Louisville Chapter); Digital Games Research Association (DiGRA); International Journal of Intelligent; Games and Simulation (IJIGS); Institute of Gaming and Animation; **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computer Science, University of Western Ontario, London, ON, Canada
Abstract: Auto dynamic difficulty (ADD) is the technique of automatically changing the level of difficulty of a video game in real time to match player expertise. Recreating an ADD system on a game-by-game basis is both expensive and time consuming; ultimately limiting its usefulness. Thus, we leverage the benefits of software design patterns to construct an ADD framework. In this paper, we demonstrate that the usage of these design patterns and this framework results in a reusable approach, both in terms of source code and process, based on our experiences with the commercial sandbox game Minecraft. We also discuss the benefits of adopting such an approach in terms of improved software quality and optimized development practices. © 2013 IEEE. (28 refs)
Main heading: Animation
Controlled terms: Computer games - Computer software selection and evaluation - Human computer interaction - Interactive computer graphics - Interactive computer systems - Multimedia systems - Software design
Uncontrolled terms: auto dynamic difficulty - Game balancing - Software design patterns - subjective difficulty - Video game
Classification Code: 722.2 Computer Peripheral Equipment - 722.4 Digital Computers and Systems - 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

43. Automated quality defect detection in software development documents

Dautovic, Andreas (1); Plösch, Reinhold (1); Saft, Matthias (2)
Source: *CEUR Workshop Proceedings*, v 708, p 29-37, 2011, *Joint Proc. of the 1st Int. Workshop on Model-Driven Software Migration, MDSM 2011 and the 5th Int. Workshop on Softw. Quality and Maintainability, SQM 2011 - Workshops at the 15th European CSMR 2011*; **ISSN:** 16130073; **Conference:** Joint 1st Int. Workshop on Model-Driven Software Migration, MDSM 2011 and the 5th International Workshop on Software Quality and Maintainability, SQM 2011 - Workshops at the 15th European Conf. on Software Maintenance and Reengineering, CSMR 2011, March 1, 2011 - March 1, 2011; **Sponsor:** Software Improvement Group (SIG); **Publisher:** Sun SITE Central Europe CEUR-WS
Author affiliation: (1) Institute for Business Informatics-Software Engineering, Johannes Kepler University Linz, Altenberger Strae 69, 4040 Linz, Austria (2) Corporate Technology, Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
Abstract: Quality of software products typically has to be assured throughout the entire software development life-cycle. However, software development documents (e.g. requirements specifications, design documents, test plans) are often not as rigorously reviewed as source code, although their quality has a major impact on the quality of the evolving software product. Due to the narrative nature of these documents, more formal approaches beyond software inspections are difficult to establish. This paper presents a tool-based approach that supports the software inspection process in order to determine defects of generally accepted documentation best practices in software development documents. By means of an empirical study we show, how this tool-based approach helps accelerating inspection tasks and facilitates gathering information on the quality of the inspected documents. (35 refs)
Main heading: Software testing

Controlled terms: Computer software selection and evaluation - Defects - Inspection - Product design - Software design - Tools
Uncontrolled terms: Empirical studies - Quality defects - Quality of softwares - Requirements specifications - Software development life cycle - Software inspection - Software products - Tool-based approach
Classification Code: 913.3.1 Inspection - 913.1 Production Engineering - 723.5 Computer Applications - 951 Materials Science - 723 Computer Software, Data Handling and Applications - 603 Machine Tools - 423 Non Mechanical Properties and Tests of Building Materials - 605 Small Tools and Hardware
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

44. Development of the clutch controller for the hybrid system using automatic code generation

Inagaki, Hiroyuki (1)
Source: *SAE Technical Papers*, v 2, 2013, *SAE 2013 World Congress and Exhibition*; **DOI:** 10.4271/2013-01-0438; **Conference:** SAE 2013 World Congress and Exhibition, April 16, 2013 - April 18, 2013; **Publisher:** SAE International
Author affiliation: (1) Aisin Seiki Co. Ltd, Japan
Abstract: The conventional software development methodology depends on the methodology where all the software components are manually coded, inspected, and tested on a real system. On the other hand, Model Based Development (MBD) is typically used to describe software development approaches in which models of software systems are created and systematically generated source code. In order to satisfy the requirements for production oriented code generation, it is necessary to address resource efficiency, reliability, and product process development. Therefore, we have designed software testing based on the software quality characteristics and evaluated the software quality of the output of a production automatic code generator. Furthermore, we have improved our modeling guideline and modified an automatic code generation tool. As a result, the software quality using automatic code generation and development efficiency has been improved simultaneously. In this paper, the software development of the clutch control for hybrid vehicles using automatic code generation is introduced. Within the clutch control development, in order to attain high robustness and response, two-degree-of-freedom robust control has been adopted as clutch control. The process from development of this controller to application software implementation is introduced together with the evaluation of the quality of automatically generated code. Copyright © 2013 SAE International. (3 refs)
Main heading: Quality control
Controlled terms: Automatic programming - Clutches - Computer software selection and evaluation - Exhibitions - Hybrid systems - Hybrid vehicles - Network components - Program compilers - Robust control - Software design - Software testing
Uncontrolled terms: Application softwares - Automatic code generations - Automatic code generators - Automatically generated - Model based development - Software development approach - Software development methodologies - Software quality characteristics
Classification Code: 913.3 Quality Assurance and Control - 902.2 Codes and Standards - 731 Automatic Control Principles and Applications - 921 Mathematics - 723 Computer Software, Data Handling and Applications - 602.2 Mechanical Transmissions - 432 Highway Transportation - 703.1 Electric Networks
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

45. Attractiveness of free and open source projects

Santos Jr., Carlos Denner (1); Pearson, John (2); Kon, Fabio (1)
Source: *18th European Conference on Information Systems, ECIS 2010*, 2010, *18th European Conference on Information Systems, ECIS 2010*; **ISBN-13:** 9780620471725; **Conference:** 18th European Conference on Information Systems, ECIS 2010, June 7, 2010 - June 9, 2010; **Publisher:** Association for Information Systems
Author affiliation: (1) University of São Paulo, Brazil (2) Southern Illinois University, Carbondale, United States
Abstract: Organisations and individuals release source code on the Web to improve their software by attracting peers in the strategic move of " opensourcing" that has created thousands of open source projects (e.g., Eclipse-IBM, Thunderbird-Mozilla and Linux-Torvalds). Nevertheless, most of these projects fail to attract people and never become active. To minimize this problem, we developed a theoretical model around a crucial construct (attractiveness) to open source projects, proposing its causes (project characteristics), indicators (e.g., number of members) and consequences (levels of activeness, efficiency, likelihood of task completion, time for task completion and software quality). We tested this model empirically using 3 samples of over 4600 projects each in a multi-sample SEM analysis. The results confirm the central role that attractiveness plays to guarantee an active and efficient community of software

development, shedding new light on whether more developers increase software quality by finding and fixing more bugs and providing upgrades. They also clarify the actual causal structure involving Web page visits, downloads and members, which can be easily mistaken. Moreover, the results can provide useful insights to strategists as we discuss the impacts of license restrictiveness, software development status, type of project and intended audience on attractiveness and its consequences. (28 refs)
Main heading: Open systems
Controlled terms: Computer operating systems - Computer software selection and evaluation - Information systems - Software design - Software engineering - World Wide Web
Uncontrolled terms: Attractiveness - Free and open source softwares - Open source projects - Project characteristics - Projects fail - SEM analysis - Software Quality - Source codes - Theoretical models
Classification Code: 903.2 Information Dissemination - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 722 Computer Systems and Equipment - 718 Telephone Systems and Related Technologies; Line Communications - 717 Optical Communication - 716 Telecommunication; Radar, Radio and Television
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

46. A study of automatic code generation

Liao, Haode (1); Jiang, Jun (1); Zhang, Yuxin (1)
Source: *Proceedings - 2010 International Conference on Computational and Information Sciences, ICCIS 2010*, p 689-691, 2010, *Proceedings - 2010 International Conference on Computational and Information Sciences, ICCIS 2010*; **ISBN-13:** 9780769542706; **DOI:** 10.1109/ICCIS.2010.171; **Article number:** 5709179; **Conference:** 2010 International Conference on Computational and Information Sciences, ICCIS2010, December 17, 2010 - December 19, 2010; **Publisher:** IEEE Computer Society
Author affiliation: (1) School of Computer Science, Southwest Petroleum University, Chengdu, Sichuan 610500, China
Abstract: Automatic code generation is the study of generative programming in the sense that the source code is generated automatically. In this paper, an approach based on component techniques that can produce in a systematic way correct, compatible and efficient database structures and manipulation function modules from abstract models is proposed. In contrast to some conventional software engineering methods, this approach has certain merits of improving software quality and shortening the software development cycle. © 2010 IEEE. (8 refs)
Main heading: Automatic programming
Controlled terms: Computer software selection and evaluation - Information science - Network components - Software design
Uncontrolled terms: Abstract models - Automatic code generations - Component techniques - Database structures - Generative programming - Manipulation functions - Software development cycles - Software engineering methods - Software Quality - Source codes
Classification Code: 703.1 Electric Networks - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 903 Information Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

47. Clone detection: Why, what and how?

Akhin, Marat (1); Itsykson, Vladimir (1)
Source: *2010 6th Central and Eastern European Software Engineering Conference, CEE-SECR 2010*, p 36-42, 2010, *2010 6th Central and Eastern European Software Engineering Conference, CEE-SECR 2010*; **Language:** Russian; **ISBN-13:** 9781457706066; **DOI:** 10.1109/CEE-SECR.2010.5783148; **Article number:** 5783148; **Conference:** 2010 6th Central and Eastern European Software Engineering Conference, CEE-SECR 2010, October 13, 2010 - October 15, 2010; **Publisher:** IEEE Computer Society
Author affiliation: (1) Saint-Petersburg State Polytechnical University, Russia
Abstract: Excessive code duplication is a bane of modern software development. Several experimental studies show that on average 15 percent of a software system can contain source code clones - repeatedly reused fragments of similar code. While code duplication may increase the speed of initial software development, it undoubtedly leads to problems during software maintenance and support. That is why many developers agree that software clones should be detected and dealt with at every stage of software development life cycle. This paper is a brief survey of current state-of-the-art in clone detection. First, we highlight main sources of code cloning such as copy-and-paste

programming, mental code patterns and performance optimizations. We discuss reasons behind the use of these techniques from the developer's point of view and possible alternatives to them. Second, we outline major negative effects that clones have on software development. The most serious drawback duplicated code have on software maintenance is increasing the cost of modifications ? any modification that changes cloned code must be propagated to every clone instance in the program. Software clones may also create new software bugs when a programmer makes some mistakes during code copying and modification. Increase of source code size due to duplication leads to additional difficulty of code comprehension. Third, we review existing clone detection techniques. Classification based on used source code representation model is given in this work. We also describe and analyze some concrete examples of clone detection techniques highlighting main distinctive features and problems that are present in practical clone detection. Finally, we point out some open problems in the area of clone detection. Currently questions like "What is a code clone?", "Can we predict the impact clones have on software quality" and "How can we increase both clone detection precision and recall at the same time?" stay open to further research. We list the most important questions in modern clone detection and explain why they continue to remain unanswered despite all the progress in clone detection research. © 2010 IEEE. (21 refs)
Main heading: Cloning
Controlled terms: Computer software maintenance - Computer software selection and evaluation - Copying - Program debugging - Quality assurance - Software design
Uncontrolled terms: Clone detection - Clone detection techniques - Code clone - Code cloning - Code comprehension - Code copying - Code duplication - Code-patterns - Copy-and-paste programming - Distinctive features - Experimental studies - Open problems - overview - Performance optimizations - Precision and recall - Program analysis - Software bug - Software clones - Software development life cycle - Software Quality - Software systems - Source codes
Classification Code: 461.8.1 Genetic Engineering - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 903.2 Information Dissemination - 913.3 Quality Assurance and Control
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

48. Assessing the precision of FindBugs by mining Java projects developed at a University

Vetro', Antonio (1); Torchiano, Marco (1); Morisio, Maurizio (1)
Source: *Proceedings - International Conference on Software Engineering*, p 110-113, 2010, *Proceedings of the 2010 7th IEEE Working Conference on Mining Software Repositories, MSR 2010, Co-located with ICSE 2010*; **ISSN:** 02705257; **ISBN-13:** 9781424468034; **DOI:** 10.1109/MSR.2010.5463283; **Article number:** 5463283; **Conference:** 7th IEEE Working Conference on Mining Software Repositories, MSR 2010, Co-located with the 2010 ACM/IEEE International Conference on Software Engineering, ICSE 2010, May 2, 2010 - May 3, 2010; **Publisher:** IEEE Computer Society
Author affiliation: (1) Politecnico di Torino, Torino, Italy
Abstract: Software repositories are analyzed to extract useful information on software characteristics. One of them is external quality. A technique used to increase software quality is automatic static analysis, by means of bug finding tools. These tools promise to speed up the verification of source code; anyway, there are still many problems, especially the high number of false positives, that hinder their large adoption in software development industry. We studied the capability of a popular bug-finding tool, FindBugs, for defect prediction purposes, analyzing the issues revealed on a repository of university Java projects. Particularly, we focused on the percentage of them that indicates actual defects with respect to their category and priority, and we ranked them. We found that a very limited set of issues have high precision and therefore have a positive impact on code external quality. © 2010 IEEE. (16 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Defects - Forecasting - Quality control - Software design - Verification
Uncontrolled terms: Automatic static analysis - Bug finding tools - Bug-finding tool - Defect prediction - External quality - False positive - High precision - Software characteristic - Software development - Software Quality - Software repositories - Source codes - Speed-ups - Static code analysis
Classification Code: 951 Materials Science - 933.1 Crystalline Solids - 922.2 Mathematical Statistics - 921 Mathematics - 913.3 Quality Assurance and Control - 912.2 Management - 723.5 Computer Applications - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 721.2 Logic Elements - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 531.2 Metallography - 423 Non Mechanical Properties and Tests of Building Materials
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

49. Toward the use of automated static analysis alerts for early identification of vulnerability- and attack-prone components

Gegick, Michael (1); Williams, Laurie (1)
Source: *Second International Conference on Internet Monitoring and Protection, ICIMP 2007*, 2007, *Second International Conference on Internet Monitoring and Protection, ICIMP 2007*; **ISBN-10:** 0769529119, **ISBN-13:** 9780769529110; **DOI:** 10.1109/ICIMP.2007.46; **Article number:** 4271764; **Conference:** 2nd International Conference on Internet Monitoring and Protection, ICIMP 2007, July 1, 2007 - July 5, 2007; **Publisher:** Inst. of Elec. and Elec. Eng. Computer Society
Author affiliation: (1) Department of Computer Science, North Carolina State University
Abstract: Extensive research has shown that software metrics can be used to identify fault- and failure-prone components. These metrics can also give early indications of overall software quality. We seek to parallel the identification and prediction of fault- and failure-prone components in the reliability context with vulnerability- and attack-prone components in the security context. Our research will correlate the quantity and severity of alerts generated by source code static analyzers to vulnerabilities discovered by manual analyses and testing. A strong correlation may indicate that automated static analyzers (ASA), a potentially early technique for vulnerability identification in the development phase, can identify high risk areas in the software system. Based on the alerts, we may be able to predict the presence of more complex and abstract vulnerabilities involved with the design and operation of the software system. An early knowledge of vulnerability can allow software engineers to make informed risk management decisions and prioritize redesign, inspection, and testing efforts. This paper presents our research objective and methodology. © 2007 IEEE. (43 refs)
Main heading: Fault tolerant computer systems
Controlled terms: Automation - Computer crime - Identification (control systems) - Risk analysis - Software design - Static analysis
Uncontrolled terms: Automated static analyzers (ASA) - Software engineers - Software metrics - Software quality
Classification Code: 722.4 Digital Computers and Systems - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.5 Computer Applications - 731.1 Control Systems - 922 Statistical Methods
Treatment: Theoretical (THR)
Database: Compendex

50. Software clustering using automated feature subset selection

Shah, Zubair (1); Naseem, Rashid (2); Orgun, Mehmet A. (3); Mahmood, Abdun (4); Shahzad, Sara (5)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 8347 LNAI, n PART 2, p 47-58, 2013, *Advanced Data Mining and Applications - 9th International Conference, ADMA 2013, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-13:** 9783642539169; **DOI:** 10.1007/978-3-642-53917-6_5; **Conference:** 9th International Conference on Advanced Data Mining and Applications, ADMA 2013, December 14, 2013 - December 16, 2013; **Sponsor:** Zhejiang University, College of Computer; Science and Technology; Taizhou University, College of Mathematics; and Information Engineering; University of Technology Sydney, Advanced; Analytics Institute; **Publisher:** Springer Verlag
Author affiliation: (1) Dept. of Computer Science, University of Venice, Italy (2) Dept. of Computer Science, City University of Science and I.T., Pakistan (3) Department of Computing, Macquarie University, Sydney, NSW, Australia (4) University of New South Wales, Canberra, ACT, Australia (5) Department of Computer Science, University of Peshawar, Pakistan
Abstract: This paper proposes a feature selection technique for software clustering which can be used in the architecture recovery of software systems. The recovered architecture can then be used in the subsequent phases of software maintenance, reuse and re-engineering. A number of diverse features could be extracted from the source code of software systems, however, some of the extracted features may have less information to use for calculating the entities, which result in dropping the quality of software clusters. Therefore, further research is required to select those features which have high relevancy in finding associations between entities. In this article first we propose a supervised feature selection technique for unlabeled data, and then we apply this technique for software clustering. A number of feature subset selection techniques in software architecture recovery have been proposed. However none of them focus on automated feature selection in this domain. Experimental results on three software test systems reveal that our proposed approach produces results which are closer to the decompositions prepared by human experts, as compared to those discovered by the well-known K-Means algorithm. © 2013 Springer-Verlag. (32 refs)
Main heading: Feature extraction
Controlled terms: Computer software reusability - Recovery
Uncontrolled terms: Architecture recovery - Automated features - Feature subset selection - K-means - Quality of softwares - Selection techniques - Software architecture recovery - Software clustering

Classification Code: 531 Metallurgy and Metallography - 716 Telecommunication; Radar, Radio and Television - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

51. More discipline needed in software engineering

Skinner, Chris (1)
Source: *Engineers Australia*, v 77, n 5, p 28-30, May 2005; **ISSN:** 14484951; **Publisher:** Institution of Engineers (Australia)
Author affiliation: (1) Display Pty Ltd., Sydney
Abstract: The various reasons as to why software engineers face problems due to slow process of recognizing software engineering as a formal engineering discipline and an overview of 2005 software engineering conference in Australia, are discussed. Software is typically measured at the source code level at which most programmers work and at which software engineers require familiarity to build the tools to create and convert software to an executable form. Software complexity is a function of many factors such as size of the software programs and information models that are involved. Engineers Australia's ITEE College Board has also proposed the formation of a National Committee of Software Engineering with terms of reference.
Main heading: Computer aided software engineering
Controlled terms: Accreditation - Computer architecture - Computer programming - Computer programming languages - Computer science - Computer software - Engineers - Information technology - Mathematical models - Semantics - Signal processing
Uncontrolled terms: Digital data - Power consumption - Software architecture - Software quality accreditation
Classification Code: 912.4 Personnel - 903.2 Information Dissemination - 901.2 Education - 901.1 Engineering Professional Aspects - 921 Mathematics - 723.5 Computer Applications - 723.1 Computer Programming - 722 Computer Systems and Equipment - 716.1 Information Theory and Signal Processing - 723.1.1 Computer Programming Languages
Treatment: General review (GEN)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

52. Software development risk model

Fawcett, James W. (1); Gungor, Murat K. (1)
Source: *Proceedings of the 2005 International Conference on Software Engineering Research and Practice, SERP'05*, v 2, p 640-645, 2005, *Proceedings of the 2005 International Conference on Software Engineering Research and Practice, SERP'05*; **ISBN-13:** 9781932415506; **Conference:** 2005 International Conference on Software Engineering Research and Practice, SERP'05, June 27, 2005 - June 30, 2005; **Publisher:** CSREA Press
Author affiliation: (1) Electrical Engineering and Computer Science Department, Syracuse University, Syracuse, NY 13244, United States
Abstract: Development of large software systems creates many, often thousands, of source code files with complex inter-dependencies. Clusters of mutually dependent files introduce the possibility of a chain of forced consequential changes when a single cluster member file is changed. Our software development risk model shows that density of dependencies within such clusters plays a crucial role in this behavior. We develop a file-rank procedure which orders the entire system's file set by increasing risk. This ranking process should prove to be useful while managing the development of large systems, indicating where attention should be focused to improve Test Risk. We have applied this model to a library from the 1.4.1 release of the open source Mozilla project with interesting results. (9 refs)
Main heading: Risk assessment
Controlled terms: Computer software selection and evaluation - Engineering research - Quality control - Risk analysis - Safety factor - Software design
Uncontrolled terms: Dependency analysis - Entire systems - Inter-dependencies - Large software systems - Large systems - Metrics - Mozilla - Open-source - Ranking process - Software development - Software quality - Source codes
Classification Code: 912 Industrial Engineering and Management - 912.2 Management - 913.3 Quality Assurance and Control - 914 Safety Engineering - 914.1 Accidents and Accident Prevention - 922 Statistical Methods - 922.1 Probability Theory - 911 Cost and Value Engineering; Industrial Economics - 408.1 Structural Design, General - 652.1 Aircraft, General - 662.1 Automobiles - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.5 Computer Applications - 901.3 Engineering Research
Database: Compendex

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

53. The design of an automated test code generation system for SQL stored procedures

Fix, Gary (1)
Source: *Proceedings - 2011 8th International Conference on Information Technology: New Generations, ITNG 2011*, p 286-290, 2010, *Proceedings - 2011 8th International Conference on Information Technology: New Generations, ITNG 2011*; **ISBN-13:** 9780769543673; **DOI:** 10.1109/ITNG.2011.57; **Article number:** 5945248; **Conference:** 2011 8th International Conference on Information Technology: New Generations, ITNG 2011, April 11, 2011 - April 13, 2011; **Sponsor:** Premier Hall for Science and Engineering (PHASE); **Publisher:** IEEE Computer Society
Author affiliation: (1) Microsoft Corporation, Redmond, WA 98052, United States
Abstract: This paper presents an overview of the design and implementation of a framework for automated test code generation of SQL stored procedures in a.NET managed code environment, using SQL, XML and the C# programming language. The primary knowledge and coding skills required for developing and using the framework are SQL, XML, and C#. The framework is most useful in a software development scenario where dedicated software test engineers are employed for testing activities, and these test engineers have SQL and XMLknowledge but not necessarily strong C# coding ability. An advantage of the framework compared to some alternative approaches is that because the framework uses SQL metadata, access to the source code of the stored procedures under test is not required. After initial development, most of the testing update effort consists of editing a relatively simple XML file. © 2011 IEEE. (5 refs)
Main heading: Software testing
Controlled terms: Ability testing - Automatic programming - Automatic testing - Computer software selection and evaluation - Engineers - Information technology - Metadata - Network components - Software design - XML
Uncontrolled terms: Alternative approach - Automated test - C# programming - Coding skills - programming environments - software quality - Software test - Source codes - stored environments - stored procedures - Test engineers - XML files
Classification Code: 422 Strength of Building Materials; Test Equipment and Methods - 423 Non Mechanical Properties and Tests of Building Materials - 703.1 Electric Networks - 723 Computer Software, Data Handling and Applications - 912.4 Personnel
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

54. Bad smells - Humans as code critics

Mäntylä, Mika V. (1); Vanhanen, Jari (1); Lassenius, Casper (1)
Source: *IEEE International Conference on Software Maintenance, ICSM*, p 399-408, 2004, *Proceedings - 20th IEEE International Conference on Software Maintenance, ICSM 2004*; **Conference:** Proceedings - 20th IEEE International Conference on Software Maintenance, ICSM 2004, September 11, 2004 - September 14, 2004; **Sponsor:** IEEE Computer Society; **Publisher:** IEEE Computer Society
Author affiliation: (1) Helsinki University of Technology, Soft. Business Engineering Institute, P.O. Box 9210, FIN-02015 HUT, Finland
Abstract: This paper presents the results of an initial empirical study on the subjective evaluation of bad code smells, which identify poor structures in software. Based on a case study in a Finnish software product company, we make two contributions. First, we studied the evaluator effect when subjectively evaluating the existence of smells in code modules. We found that the use of smells for code evaluation purposes is hard due to conflicting perceptions of different evaluators. Second, we applied source code metrics for identifying three smells and compared these results to the subjective evaluations. Surprisingly, the metrics and smell evaluations did not correlate. © 2004 IEEE. (40 refs)
Main heading: Software engineering
Controlled terms: Algorithms - Codes (symbols) - Computer software maintenance - Human engineering - Quality assurance
Uncontrolled terms: Bad code smells - Software design - Software evolution - Software quality
Classification Code: 461.4 Ergonomics and Human Factors Engineering - 723.1 Computer Programming - 723.2 Data Processing and Image Processing - 913.3 Quality Assurance and Control
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

55. THEX: Mining metapatterns from java

Posnett, Daryl (1); Bird, Christian (1); Devanbu, Premkumar (1)
Source: *Proceedings - International Conference on Software Engineering*, p 122-125, 2010, *Proceedings of the 2010 7th IEEE Working Conference on Mining Software Repositories, MSR 2010, Co-located with ICSE 2010*; **ISSN:** 02705257; **ISBN-13:** 9781424468034; **DOI:** 10.1109/MSR.2010.5463349; **Article number:** 5463349; **Conference:** 7th IEEE Working Conference on Mining Software Repositories, MSR 2010, Co-located with the 2010 ACM/IEEE International Conference on Software Engineering, ICSE 2010, May 2, 2010 - May 3, 2010; **Publisher:** IEEE Computer Society
Author affiliation: (1) Department of Computer Science, University of California, Davis, CA, United States
Abstract: Design patterns are codified solutions to common object-oriented design (OOD) problems in software development. One of the proclaimed benefits of the use of design patterns is that they decouple functionality and enable different parts of a system to change frequently without undue disruption throughout the system. These OOD patterns have received a wealth of attention in the research community since their introduction; however, identifying them in source code is a difficult problem. In contrast, metapatterns have similar effects on software design by enabling portions of the system to be extended or modified easily, but are purely structural in nature, and thus easier to detect. Our long-term goal is to evaluate the effects of different OOD patterns on coordination in software teams as well as outcomes such as developer productivity and software quality. we present THEX, a metapattern detector that scales to large codebases and works on any Java bytecode. We evaluate THEX by examining its performance on codebases with known design patterns (and therefore metapatterns) and find that it performs quite well, with recall of over 90%. © 2010 IEEE. (11 refs)
Main heading: Software design
Controlled terms: Computer software selection and evaluation - Design - Java programming language - Object oriented programming
Uncontrolled terms: Design Patterns - Java byte codes - Long-term goals - Object-oriented design - Research communities - Software development - Software Quality - Software teams - Source codes
Classification Code: 912.2 Management - 902.1 Engineering Graphics - 723.5 Computer Applications - 723.1.1 Computer Programming Languages - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 408 Structural Design
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

56. A study on compiler selection in safety-critical redundant system based on airworthiness requirement

Wei, Chang (1); Xiaohong, Bao (1); Tingdi, Zhao (1)
Source: *Procedia Engineering*, v 17, p 497-504, 2011, *Proceedings of 2nd International Symposium on Aircraft Airworthiness, ISAA 2011*; **ISSN:** 18777058; **DOI:** 10.1016/j.proeng.2011.10.060; **Conference:** 2nd International Symposium on Aircraft Airworthiness, ISAA 2011, October 26, 2011 - October 28, 2011; **Sponsor:** Beihang University; National Laboratory for Aeronautics and Astronautics (NLAA); Aircr. Airworthiness Certif. Dep., Civ. Aviat. Adm. China (CAAC); **Publisher:** Elsevier Ltd
Author affiliation: (1) School of Reliability and System Engineering, Beihang University, Beijing, 100191, China
Abstract: The dependability of compiler would directly affect the quality of software because it can directly produce object code. At the same time, compiler diversity is an important part of software diversity design in a redundant system, which could not only help avoid common defects from compilers but also to find defects in source code. This paper proposes a method for compiler selection in safety-critical embedded redundant system based on airworthiness requirement and the principle of software diversity. A case on compiler selection in tri-redundancy FCS (flight control system) is given in the end. (21 refs)
Main heading: Program compilers
Controlled terms: Codes (symbols) - Defects - Embedded software - Flight control systems - Safety engineering - Software design
Uncontrolled terms: airworthiness requirement - compiler dependability - compiler diversity - compiler selection - Redundant system
Classification Code: 423 Non Mechanical Properties and Tests of Building Materials - 723 Computer Software, Data Handling and Applications - 731.1 Control Systems - 914 Safety Engineering - 951 Materials Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

57. Modular specification and checking of structural dependencies

Mitschke, Ralf (1); Eichberg, Michael (1); Mezini, Mira (1); Garcia, Alessandro (2); Macia, Isela (2)
Source: *AOSD 2013 - Proceedings of the 2013 ACM on Aspect-Oriented Software Development*, p 85-96, 2013, *AOSD 2013 - Proceedings of the 2013 ACM on Aspect-Oriented Software Development*; **ISBN-13:** 9781450317665; **DOI:** 10.1145/2451436.2451448; **Conference:** 12th Annual International Conference on Aspect-Oriented Software Development, AOSD 2013, March 24, 2013 - March 29, 2013; **Sponsor:** Aspect-Oriented Software Association (AOSA); **Publisher:** Association for Computing Machinery
Author affiliation: (1) Technische Universität Darmstadt, Darmstadt, Germany (2) Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
Abstract: Checking a software's structural dependencies is a line of research on methods and tools for analyzing, modeling and checking the conformance of source code w.r.t. specifications of its intended static structure. Existing approaches have focused on the correctness of the specification, the impact of the approaches on software quality and the expressiveness of the modeling languages. However, large specifications become unmaintainable in the event of evolution without the means to modularize such specifications. We present Vespucci, a novel approach and tool that partitions a specification of the expected and allowed dependencies into a set of cohesive slices. This facilitates modular reasoning and helps individual maintenance of each slice. Our approach is suited for modeling high-level as well as detailed low-level decisions related to the static structure and combines both in a single modeling formalism. To evaluate our approach we conducted an extensive study spanning nine years of the evolution of the architecture of the object-relational map- ping framework Hibernate. Copyright © 2013 ACM. (33 refs)
Main heading: Specifications
Controlled terms: Computer software selection and evaluation - Computer systems programming - Scalability - Software architecture - Static analysis
Uncontrolled terms: Dependency constraints - Modeling formalisms - Modeling languages - Modular reasoning - Modular specifications - Modularity - Object-relational - Static structures
Classification Code: 718 Telephone Systems and Related Technologies; Line Communications - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 902.2 Codes and Standards - 961 Systems Science
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

58. Architecture reconstruction and analysis of medical device software

Ganesan, Dharmalingam (1); Lindvall, Mikael (1); Cleaveland, Rance (1); Jetley, Raoul (2); Jones, Paul (2); Zhang, Yi (2)
Source: *Proceedings - 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011*, p 194-203, 2011, *Proceedings - 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011*; **ISBN-13:** 9780769543512; **DOI:** 10.1109/WICSA.2011.33; **Article number:** 5959691; **Conference:** 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011, June 20, 2011 - June 24, 2011; **Sponsor:** IBM; IEEE Computer Society; IFIP; Software Engineering Institute, Carnegie Mellon; SIEMENS; **Publisher:** IEEE Computer Society
Author affiliation: (1) Fraunhofer CESE, College Park, MD, United States (2) FDA, Silver Spring, MD, United States
Abstract: New research is underway at the FDA to investigate the benefits of integrating architecture analysis into safety evaluations of medical-device software. Due to the complexity in setting up testing environments for such software, the FDA is unable to conduct large-scale safety testing; instead, it must rely on other techniques to build an argument for whether the software is safe or not. The architecture analysis approach, formalized using relational algebra, is based on reconstructing abstract, yet precise, architectural views from source code to help build such arguments about safety. This paper discusses the use of the formal approach to analyze the Computer-Assisted Resuscitation Algorithm (CARA) software, which controls an infusion pump designed to provide automated assistance for transfusing blood. The results suggest that a) architecture analysis offers many insights related to software quality in general and testability (i.e., the ease of testing) and its impact on safety in particular, and b) architectural analysis results can be used to help configure static analysis tools to improve their performance for verifying safety properties. © 2011 IEEE. (30 refs)
Main heading: Software architecture
Controlled terms: Biomedical engineering - Computer software selection and evaluation - Occupational risks - Resuscitation - Safety testing - Software testing - Static analysis
Uncontrolled terms: Architectural analysis - Architectural views - Architecture analysis - Architecture reconstruction - Automated assistance - Computer assisted - Formal approach - Infusion pump - Medical device safety - Medical Devices - Relational algebra - Reverse architecting - Safety evaluations - Safety property - Software Quality - Source codes - Testability - Testing environment - Verifiability

Classification Code: 461.1 Biomedical Engineering - 461.6 Medicine and Pharmacology - 723 Computer Software, Data Handling and Applications - 723.5 Computer Applications - 914.1 Accidents and Accident Prevention
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

59. A case study of bias in bug-fix datasets

Nguyen, Thanh H.D. (1); Adams, Bram (1); Hassan, Ahmed E. (1)
Source: *Proceedings - Working Conference on Reverse Engineering, WCRE*, p 259-268, 2010, *Proceedings - 17th Working Conference on Reverse Engineering, WCRE 2010*; **ISSN:** 10951350; **ISBN-13:** 9780769541235; **DOI:** 10.1109/WCRE.2010.37; **Article number:** 5645567; **Conference:** 17th Working Conference on Reverse Engineering, WCRE 2010, October 13, 2010 - October 16, 2010; **Sponsor:** Reengineering Forum; **Publisher:** IEEE Computer Society
Author affiliation: (1) Software Analysis and Intelligence Lab. (SAIL), School of Computing, Queen's University, Kingston, ON, Canada
Abstract: Software quality researchers build software quality models by recovering traceability links between bug reports in issue tracking repositories and source code files. However, all too often the data stored in issue tracking repositories is not explicitly tagged or linked to source code. Researchers have to resort to heuristics to tag the data (e.g., to determine if an issue is a bug report or a work item), or to link a piece of code to a particular issue or bug. Recent studies by Bird et al. and by Antoniol et al. suggest that software models based on imperfect datasets with missing links to the code and incorrect tagging of issues, exhibit biases that compromise the validity and generality of the quality models built on top of the datasets. In this study, we verify the effects of such biases for a commercial project that enforces strict development guidelines and rules on the quality of the data in its issue tracking repository. Our results show that even in such a perfect setting, with a near-ideal dataset, biases do exist - leading us to conjecture that biases are more likely a symptom of the underlying software development process instead of being due to the used heuristics. © 2010 IEEE. (16 refs)
Main heading: Computer software selection and evaluation
Controlled terms: Research - Reverse engineering - Software design
Uncontrolled terms: Bias - Bug-fix - Data quality - Prediction - Sample
Classification Code: 723 Computer Software, Data Handling and Applications - 901.3 Engineering Research
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

60. Supporting software decision meetings: Heatmaps for visualising test and code measurements

Feldt, Robert (1); Staron, Miroslaw (1); Hult, Erika (2); Liljegren, Thomas (2)
Source: *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*, p 62-69, 2013, *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*; **ISBN-13:** 9780769550916; **DOI:** 10.1109/SEAA.2013.61; **Article number:** 6619490; **Conference:** 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013, September 4, 2013 - September 6, 2013; **Publisher:** IEEE Computer Society
Author affiliation: (1) Computer Science and Engineering, Chalmers and University of Gothenburg, Sweden (2) RUAG Space, Gothanburg, Sweden
Abstract: To achieve software quality it is critical to quickly understand the current test status, its changes over time as well as its relation to source code changes. However, even if this information is available in test logs and code repositories it is seldomly put to good use in supporting decision processes in software development. The amount of information is often large, is time consuming to extract and hard to monitor. This case study shows how visualisation and correlation between software measurements can support improvement discussions. In particular, simple heat maps were found to be effective to visualize and monitor changes and identify recurring patterns in the development of a space-bourn, embedded control system. Statistical analysis quantified the correlation between different sources of development data and heat maps then effectively focused the attention of stakeholders to importants parts of the system. Here the visual analysis was focused on post-project, historical data but we discuss how early identification based on dynamic data analysis could support more effective analysis, planning and execution of quality assurance. Based on our findings we state requirements on such an online, visual analysis system and present a prototype implementation that can help software measurements better support value-based decisions in software development. © 2013 IEEE. (19 refs)
Main heading: Software testing

Controlled terms: Computer software selection and evaluation - Project management - Quality assurance - Software design - Visualization
Uncontrolled terms: Amount of information - Embedded control systems - Empirical studies - Industry case studies - Planning and execution - Prototype implementations - Software Measurement - Software project management
Classification Code: 723.5 Computer Applications - 902.1 Engineering Graphics - 912.2 Management - 913.3 Quality Assurance and Control
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

61. Monitoring code quality and development activity by software maps

Bohnet, Johannes (1); Döllner, Jürgen (1)
Source: *Proceedings - International Conference on Software Engineering*, p 9-16, 2011, *MTD'11 - Proceedings of the 2nd Workshop on Managing Technical Debt, Co-located with ICSE 2011*; **ISSN:** 02705257; **ISBN-13:** 9781450305860; **DOI:** 10.1145/1985362.1985365; **Conference:** 2nd International Workshop on Managing Technical Debt, MTD 2011, Co-located with the 33rd International Conference on Software Engineering, ICSE 2011, May 23, 2011 - May 23, 2011; **Sponsor:** ACM SIGSOFT; IEEE CS; **Publisher:** IEEE Computer Society
Author affiliation: (1) Hasso-Plattner-Institute for IT Systems Engineering, University of Potsdam, Potsdam, Germany
Abstract: Software development projects are difficult to manage, in general, due to the friction between completing system features and, at the same time, obtaining a high degree of code quality to ensure maintainability of the system in the future. A major challenge of this optimization problem is that code quality is less visible to stakeholders in the development process, particularly, to the management. In this paper, we describe an approach for automated software analysis and monitoring of both quality-related code metrics and development activities by means of software maps. A software map represents an adaptive, hierarchical representation of software implementation artifacts such as source code files being organized in a modular hierarchy. The maps can express and combine information about software development, software quality, and system dynamics; they can systematically be specified, automatically generated, and organized by templates. The maps aim at supporting decision-making processes. For example, they facilitate to decide where in the code an increase of quality would be beneficial both for speeding up current development activities and for reducing risks of future maintenance problems. Due to their high degree of expressiveness and their instantaneous generation, the maps additionally serve as up-to-date information tools, bridging an essential information gap between management and development, improve awareness, and serve as early risk detection instrument. The software map concept and its tool implementation are evaluated by means of two case studies on large industrially developed software systems. © 2011 ACM. (12 refs)
Main heading: Software design
Controlled terms: Computer software maintenance - Computer software selection and evaluation - Decision making - Maintainability - Quality control - Visualization
Uncontrolled terms: managing technical debt - Refactorings - Software analysis - software quality - software visualization
Classification Code: 723 Computer Software, Data Handling and Applications - 902.1 Engineering Graphics - 912.2 Management - 913.3 Quality Assurance and Control - 913.5 Maintenance
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

62. Semantic software metrics computed from natural language design specifications

Gall, C.S. (1); Lukins, S. (2); Etzkorn, L. (2); Gholston, S. (3); Farrington, P. (3); Utley, D. (3); Fortune, J. (3); Virani, S. (3)
Source: *IET Software*, v 2, n 1, p 17-26, 2008; **ISSN:** 17518806; **DOI:** 10.1049/iet-sen:20070109; **Publisher:** Institution of Engineering and Technology
Author affiliation: (1) University of Alabama in Huntsville, Information Technology and Systems Center, Huntsville, AL 35899, United States (2) University of Alabama in Huntsville, Computer Science Department, Huntsville, AL 35899, United States (3) University of Alabama in Huntsville, Industrial and Systems Engineering and Engineering Management Department, Huntsville, AL 35899, United States
Abstract: An approach using semantic metrics to provide insight into software quality early in the design phase of software development by automatically analysing natural language (NL) design specifications for object-oriented systems is presented. Semantic metrics are based on the meaning of software within the problem domain. In this paper, we extend semantic metrics to analyse design specifications. Since semantic metrics can now be calculated from early in design through software maintenance, they provide a consistent and seamless type of metric that can

be collected through the entire lifecycle. We discuss our semMet system, an NL-based program comprehension tool we have expanded to calculate semantic metrics from design specifications. To validate semantic metrics from design specifications and to illustrate their seamless nature across the software lifecycle, we compare semantic metrics from different phases of the lifecycle, and we also compare them to syntactically oriented metrics calculated from the source code. Results indicate semantic metrics calculated from design specifications can give insight into the quality of the source code based on that design. Also, these results illustrate that semantic metrics provide a consistent and seamless type of metric that can be collected through the entire lifecycle. © The Institution of Engineering and Technology 2008. (27 refs)
Main heading: Semantic Web
Controlled terms: Computer software maintenance - Life cycle - Natural language processing systems - Software design
Uncontrolled terms: Design specifications - Semantic metrics
Classification Code: 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming - 723.2 Data Processing and Image Processing - 723.5 Computer Applications - 903 Information Science - 913.1 Production Engineering
Treatment: Theoretical (THR)
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

63. Software-in-the loop based end to end validation methodology for aerospace software development

Prasada, Kumari K.S. (1); Desai, Kiran (1); Dutta, Shuvo (1); Prasad, B.V. (1)
Source: *60th International Astronautical Congress 2009, IAC 2009*, v 10, p 8089-8095, 2009, *60th International Astronautical Congress 2009, IAC 2009*; **ISBN-13:** 9781615679089; **Conference:** 60th International Astronautical Congress 2009, IAC 2009, October 12, 2009 - October 16, 2009; **Publisher:** International Astronautical Federation, IAF
Author affiliation: (1) ISRO Satellite Center, Bangalore, India
Abstract: ISRO Satellite Centre (ISAC) is the lead centre of the Indian Space Research Organisation in the development and operationalisation of satellites for communication, navigation and remote sensing applications. In all these spacecrafts, highly advanced embedded systems carryout variety of mission critical functions. A typical example of such a system is the satellite Attitude and Orbit Control System -the on board computer which is the brain of the satellite. As per existing practices, testing of on board software to confirm its functioning in a simulated dynamic environment takes place only when the software is integrated with OBC hardware and system level tests in closed loop mode are conducted. On the contrary, by the new technique called the Software In Loop Simulation (SILS) test method, the on-board software can be hilly tested in a software simulated dynamic environment without OBC hardware. This method of closed loop flight software validation is demonstrated with CARTOSAT-2 AOCS software using SILS test bed. The results very well demonstrate the effectiveness of the technique in early performance prediction and assessment of flight software. This validation philosophy will be followed for future spacecraft GNC systems. In a development environments where software requirements are too complex and requirements changes are to be incorporated even during final stages of development, this technique offers an excellent solution in hilly validating on board software at source code level before it gets integrated with target hardware. This additional validation step not only improves software quality but also enhances productivity and reduces system turnaround time. (10 refs)
Main heading: Software design
Controlled terms: Communication satellites - Computer control systems - Computer hardware - Computer software selection and evaluation - Embedded systems - Equipment testing - Flight control systems - Navigation - Philosophical aspects - Remote sensing - Space research - Spacecraft - Turnaround time
Uncontrolled terms: Aerospace software - Closed loop mode - Closed loops - Development environment - Dynamic environments - End to end - Flight Software - Future spacecraft - Mission critical - On board softwares - Onboard computers - Performance prediction - Remote sensing applications - Requirements change - Satellite attitude - Software Quality - Software requirements - Source codes - System-level test - Target hardware - Test method - Validation methodologies
Classification Code: 944 Moisture, Pressure and Temperature, and Radiation Measuring Instruments - 723.1 Computer Programming - 723.5 Computer Applications - 731.1 Control Systems - 901.1 Engineering Professional Aspects - 912 Industrial Engineering and Management - 912.2 Management - 913 Production Planning and Control; Manufacturing - 941 Acoustical and Optical Measuring Instruments - 942 Electric and Electronic Measuring Instruments - 943 Mechanical and Miscellaneous Measuring Instruments - 723 Computer Software, Data Handling and Applications - 431.5 Air Navigation and Traffic Control - 434.4 Waterway Navigation - 652.3 Aircraft Instruments and Equipment - 655.1 Spacecraft, General - 722 Computer Systems and Equipment - 655.2.1 Communication Satellites - 716

Telecommunication; Radar, Radio and Television - 716.3 Radio Systems and Equipment - 717 Optical Communication - 656.2 Space Research
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

64. Tool-supported estimation of software evolution effort in service-oriented systems

Stammel, Johannes (1); Trifu, Mircea (1)
Source: *CEUR Workshop Proceedings*, v 708, p 56-63, 2011, *Joint Proc. of the 1st Int. Workshop on Model-Driven Software Migration, MDSM 2011 and the 5th Int. Workshop on Softw. Quality and Maintainability, SQM 2011 - Workshops at the 15th European CSMR 2011*; **ISSN:** 16130073; **Conference:** Joint 1st Int. Workshop on Model-Driven Software Migration, MDSM 2011 and the 5th International Workshop on Software Quality and Maintainability, SQM 2011 - Workshops at the 15th European Conf. on Software Maintenance and Reengineering, CSMR 2011, March 1, 2011 - March 1, 2011; **Sponsor:** Software Improvement Group (SIG); **Publisher:** Sun SITE Central Europe CEUR-WS
Author affiliation: (1) FZI Forschungszentrum Informatik, 10-14 Haid-und-Neu Str., Karlsruhe, Germany
Abstract: Existing software systems need to evolve in order to keep up with changes in requirements, platforms and technologies. And because software evolution is a costly business, an early and accurate estimation of evolution efforts is highly desirable in any software development project. In this paper we present KAMP, a tool-supported approach, based on change impact analysis, enabling software architects to express a potential design for a given change request at the architecture level and to accurately estimate the evolution effort associated with its implementation in source code. The approach is also used to compare alternative designs before carrying out the implementation work. We apply the KAMP approach on an enterprise SOA showcase and show how it supports the predictable and cost-effective evolution of this software system. (14 refs)
Main heading: Quality of service
Controlled terms: Architecture - Computer software selection and evaluation - Estimation - Software architecture - Tools
Uncontrolled terms: Accurate estimation - Alternative designs - Change impact analysis - Effort Estimation - Service Oriented Systems - Software architects - Software development projects - Software Evolution
Classification Code: 723 Computer Software, Data Handling and Applications - 718 Telephone Systems and Related Technologies; Line Communications - 717 Optical Communication - 921 Mathematics - 716 Telecommunication; Radar, Radio and Television - 603 Machine Tools - 402 Buildings and Towers - 605 Small Tools and Hardware
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

65. Visually localizing design problems with disharmony maps

Wettel, Richard (1); Lanza, Michele (1)
Source: *SOFTVIS 2008 - Proceedings of the 4th ACM Symposium on Software Visualization*, p 155-164, 2008, *SOFTVIS 2008 - Proceedings of the 4th ACM Symposium on Software Visualization*; **ISBN-13:** 9781605581125; **DOI:** 10.1145/1409720.1409745; **Conference:** 4th ACM Symposium on Software Visualization, SOFTVIS 2008, September 16, 2008 - September 17, 2008; **Publisher:** Association for Computing Machinery
Author affiliation: (1) REVEAL Faculty of Informatics, University of Lugano, Switzerland
Abstract: Assessing the quality of software design is difficult, as "design" is expressed through guidelines and heuristics, not rigorous rules. One successful approach to assess design quality is based on detection strategies, which are metrics-based composed logical conditions, by which design fragments with specific properties are detected in the source code. Such detection strategies, when executed on large software systems usually return large sets of artifacts, which potentially exhibit one or more "design disharmonies", which are then inspected manually, a cumbersome activity. In this article we present disharmony maps, a visualization-based approach to locate such flawed software artifacts in large systems. We display the whole system using a 3D visualization technique based on a city metaphor. We enrich such visualizations with the results returned by a number of detection strategies, and thus render both the static structure and the design problems that affect a subject system. We evaluate our approach on a number of open-source Java systems and report on our findings. © 2008 ACM. (37 refs)
Main heading: Software design
Controlled terms: Computer software selection and evaluation - Design - Three dimensional computer graphics - Visualization

Uncontrolled terms: 3-d visualizations - Design problems - Design qualities - Java systems - Large software systems - Large systems - Open sources - Quality of softwares - Software artifacts - Software visualization - Source codes - Static structures - Whole systems
Classification Code: 912.2 Management - 902.1 Engineering Graphics - 723.5 Computer Applications - 723.2 Data Processing and Image Processing - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 408 Structural Design
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

66. DeMIMA: A multilayered approach for design pattern identification

Guéhéneuc, Yann-Gaël (1); Antoniol, Giuliano (2)
Source: *IEEE Transactions on Software Engineering*, v 34, n 5, p 667-684, 2008; **ISSN:** 00985589; **DOI:** 10.1109/TSE.2008.48; **Publisher:** Institute of Electrical and Electronics Engineers Inc.
Author affiliation: (1) Département d'Informatique et Recherche Opérationnelle, Université de Montréal, C.P. 6128, succ. Centre Ville, Montréal, QC H3C 3J7, Canada (2) Département d'Informatique, École Polytechnique de Montréal, C.P. 6079, succ. Centre Ville, Montréal, QC H3C 3A7, Canada
Abstract: Design patterns are important in object-oriented programming because they offer design motifs, elegant solutions to recurrent design problems, which improve the quality of software systems. Design motifs facilitate system maintenance by helping to understand design and implementation. However, after implementation, design motifs are spread throughout the source code and are thus not directly available to maintainers. We present DeMIMA, an approach to identify semi-automatically micro-architectures that are similar to design motifs in source code and to ensure the traceability of these micro-architectures between implementation and design. DeMIMA consists of three layers: two layers to recover an abstract model of the source code, including binary class relationships, and a third layer to identify design patterns in the abstract model. We apply DeMIMA to five open-source systems and, on average, we observe 34 percent precision for the considered 12 design motifs. Through the use of explanation-based constraint programming, DeMIMA ensures 100 percent recall on the five systems. We also apply DeMIMA on 33 industrial components. © 2008 IEEE. (48 refs)
Main heading: Architectural design
Controlled terms: Abstracting - Codes (symbols) - Computer programming - Constraint theory - Design - Maintenance - Object oriented programming
Uncontrolled terms: Abstract models - Binary class relationships - Constraint programmings - Design motifs - Design patterns - Design problems - Industrial components - Interclass relationships - Multilayered approaches - Object-oriented programmings - Quality of softwares - Source codes - Source systems - System maintenances - Two layers
Classification Code: 961 Systems Science - 913.5 Maintenance - 903.1 Information Sources and Analysis - 902.1 Engineering Graphics - 731.1 Control Systems - 723.2 Data Processing and Image Processing - 723.1 Computer Programming - 721.1 Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory - 408.1 Structural Design, General - 408 Structural Design - 402 Buildings and Towers
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

67. Analysis of components for generalization using multidimensional scaling

Damaeviius, Robertas (1)
Source: *Fundamenta Informaticae*, v 91, n 3-4, p 507-522, 2009; **ISSN:** 01692968; **DOI:** 10.3233/FI-2009-0054; **Publisher:** IOS Press
Author affiliation: (1) Software Engineering Department, Kaunas University of Technology, Studentu 50-415, LT-51368 Kaunas, Lithuania
Abstract: To achieve better software quality, to shorten software development time and to lower development costs, software engineers are adopting generative reuse as a software design process. The usage of generic components allows increasing reuse and design productivity in software engineering. Generic component design requires systematic domain analysis to identify similar components as candidates for generalization. However, component feature analysis and identification of components for generalization usually is done ad hoc. In this paper, we propose to apply a data visualization method, called Multidimensional Scaling (MDS), to analyze software components in the multidimensional feature space. Multidimensional data that represent syntactical and semantic features of source code components are mapped to 2D space. The results of MDS are used to partition an initial set of components into groups of similar source code components that can be further used as candidates for generalization. STRESS value is used to

estimate the generalizability of a given set of components. Case studies for Java Buffer and Geom class libraries are presented. (46 refs)
Main heading: Software design
Controlled terms: Computer software reusability - Computer software selection and evaluation - Data visualization - Embedded systems - Engineering
Uncontrolled terms: Component based software engineering - Domain analysis - Feature engineering - Generalization - Multidimensional scaling - Software similarity
Classification Code: 912.2 Management - 901 Engineering Profession - 723.5 Computer Applications - 921.4 Combinatorial Mathematics, Includes Graph Theory, Set Theory - 723.2 Data Processing and Image Processing - 723 Computer Software, Data Handling and Applications - 722 Computer Systems and Equipment - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

68. Architecture framework for software test tool

Sun, Chang-ai (1); Liu, Chao (1); Jin, Mao-zhong (1); Zhang, Mei (1)
Source: *Proceedings of the Conference on Technology of Object-Oriented Languages and Systems, TOOLS*, n TOOL 36, p 40-47, 2000; **ISSN:** 15302067; **Conference:** 36th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS 36), October 30, 2000 - November 4, 2000; **Sponsor:** National Natural Science Foundation of China; Xian Municipal Science and Technology Commission; ISE; **Publisher:** IEEE Comp Soc
Author affiliation: (1) Beijing Univ of Aeronautics &, Astronautics, Beijing, China
Abstract: Software test tool based on source code is an important aided tool of software quality assurance, and under the environments that test technology and test requirement continuously vary, Software test tool itself should be endowed with extensibility, easy reuse and interoperation. In the paper, the necessity of research on integrated framework of component-based software test tool is analyzed, based on the analysis of traditional model of software test tool and characteristic of software test tool, we study architecture design of integrated framework of software test tool, come up with a reference model of integrated framework of software test tool, which is based on component and can be compatible with CORBA. Moreover, the technology of implementing dynamic configuration based on different user's requirements is also introduced. (9 refs)
Main heading: Object oriented programming
Controlled terms: Computer aided software engineering - Computer architecture - Computer software reusability - Interoperability
Uncontrolled terms: Common object request broker architecture (CORBA) - Domain specific modeling
Classification Code: 723 Computer Software, Data Handling and Applications - 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

69. TECDP: A tool for extracting creational design patterns

Sudhakar, N. (1); Gyani, J. (1)
Source: *ICWET 2010 - International Conference and Workshop on Emerging Trends in Technology 2010, Conference Proceedings*, p 735-736, 2010, *ICWET 2010 - International Conference and Workshop on Emerging Trends in Technology 2010, Conference Proceedings*; **ISBN-13:** 9781605588124; **DOI:** 10.1145/1741906.1742077; **Conference:** International Conference and Workshop on Emerging Trends in Technology 2010, ICWET 2010, February 26, 2010 - February 27, 2010; **Sponsor:** ACM Special Interest Group on Artificial Intelligence (SIGART); All India Council of Technical Education; Unitech Engineers; ACM Special Interest Group on Computer Architecture (SIGARCH); **Publisher:** Association for Computing Machinery
Author affiliation: (1) Jayamukhi Institute of Technological Sciences, Narsampet, Warangal.(A.P.), India
Abstract: Refactoring using design patterns leads to production of high quality and easily maintainable software. Without an acceptable level of design patterns in the development of software, it will not be able to meet the demands of software industry. Promoting design patterns requires effective support. In this paper we propose a tool to extract few creational design patterns by detecting Intent Aspects(IA's) from Java source code by applying reverse engineering algorithms. This helps in refactoring the code and thus improves the quality of software, in terms of reusability, flexibility and extendibility. Copyright 2010 ACM. (14 refs)
Main heading: Software design

Controlled terms: Computer software reusability - Computer software selection and evaluation - Java programming language - Reengineering - Reusability - Reverse engineering
Uncontrolled terms: Design Patterns - Extendibility - High quality - Intent-aspects - Java source codes - Quality of softwares - Refactorings - Software industry
Classification Code: 913.3 Quality Assurance and Control - 912.2 Management - 723.5 Computer Applications - 723.1.1 Computer Programming Languages - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 452.3 Industrial Wastes
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

70. Applying model transformations to optimizing real-time QoS configurations in DRE systems

Kavimandan, Amogh (1); Gokhale, Aniruddha (1)
Source: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v 5581 LNCS, p 18-35, 2009, *Architectures for Adaptive Software Systems - 5th International Conference on the Quality of Software Architectures, QoSA 2009, Proceedings*; **ISSN:** 03029743, **E-ISSN:** 16113349; **ISBN-10:** 3642023509, **ISBN-13:** 9783642023507; **DOI:** 10.1007/978-3-642-02351-4_2; **Conference:** 5th International Conference on the Quality of Software Architectures, QoSA 2009, June 24, 2009 - June 26, 2009; **Publisher:** Springer Verlag
Author affiliation: (1) ISIS, Dept. of EECS, Vanderbilt University, Nashville, TN, United States
Abstract: The quality of a software architecture for component-based distributed systems is defined not just by its source code but also by other systemic artifacts, such as the assembly, deployment, and configuration of the application components and their component middleware. In the context of distributed, real-time, and embedded (DRE) component-based systems, bin packing algorithms and schedulability analysis have been used to make deployment and configuration decisions. However, these algorithms make only coarse-grained node assignments but do not indicate how components are allocated to different middleware containers on the node, which are known to impact runtime system performance and resource consumption. This paper presents a model transformation-based algorithm that combines user-specified quality of service (QoS) requirements with the node assignments to provide a finer level of granularity and precision in the deployment and configuration decisions. A beneficial side effect of our work lies in how these decisions can be leveraged by additional backend performance optimization techniques. We evaluate our approach and compare it against the existing state-of-the-art in the context of a representative DRE system. © 2009 Springer Berlin Heidelberg. (27 refs)
Main heading: Software architecture
Controlled terms: Computer software selection and evaluation - Embedded systems - Middleware - Quality of service - Real time systems
Uncontrolled terms: Application components - Bin packing algorithm - Coarse-grained - Component based - Component middleware - Component-based systems - Deployment and configuration - Distributed systems - DRE systems - Graph/model transformations - Model transformation - Model-driven engineering - Performance optimizations - Resource consumption - Runtime systems - Schedulability analysis - Side effect - Source codes
Classification Code: 912.2 Management - 723.5 Computer Applications - 723.1 Computer Programming - 723 Computer Software, Data Handling and Applications - 722.4 Digital Computers and Systems - 722 Computer Systems and Equipment - 718 Telephone Systems and Related Technologies; Line Communications - 717 Optical Communication - 716 Telecommunication; Radar, Radio and Television
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

71. Mining development repositories to study the impact of collaboration on software systems

Bettenburg, Nicolas (1)
Source: *SIGSOFT/FSE 2011 - Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*, p 376-379, 2011, *SIGSOFT/FSE'11 - Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*; **ISBN-13:** 9781450304436; **DOI:** 10.1145/2025113.2025165; **Conference:** 19th ACM SIGSOFT Symposium on Foundations of Software Engineering, SIGSOFT/FSE'11, September 5, 2011 - September 9, 2011; **Sponsor:** Assoc. Comput. Mach., Spec.; Interest Group Softw. Eng. (ACM SIGSOFT); **Publisher:** Association for Computing Machinery

Author affiliation: (1) Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada
Abstract: Software development is a largely collaborative effort, of which the actual encoding of program logic in source code is a relatively small part. Yet, little is known about the impact of collaboration between stakeholders on software quality. We hypothesize that the collaboration between stakeholders during software development has a non-negligible impact on the software system. Information about collaborative activities can be recovered from traces of their communication, which are recorded in the repositories used for the development of the software system. This thesis contributes the following: 1) to make this information accessible for practitioners and researchers, we present approaches to distill communication information from development repositories, and empirically validate our proposed extractors. 2) By linking back the extracted communication data to the parts of the software system under discussion, we are able to empirically study the impact of communication, as a proxy to collaboration between stakeholders, on a software system. Through case studies on a broad spectrum of open-source software projects, we demonstrate the important role of social interactions between stakeholders with respect to the evolution of a software system. © 2011 ACM. (34 refs)
Main heading: Software design
Controlled terms: Communication - Computer software selection and evaluation - Distributed computer systems
Uncontrolled terms: Collaboration - Empirical studies - Sociotechnical - Software repositories - Unstructured data
Classification Code: 716 Telecommunication; Radar, Radio and Television - 722.4 Digital Computers and Systems - 723 Computer Software, Data Handling and Applications
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

72. An approach for collaborative code reviews using multi-touch technology

Müller, Sebastian (1); Wursch, Michael (1); Fritz, Thomas (1); Gall, Harald C. (1)
Source: *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering, CHASE 2012 - Proceedings*, p 93-99, 2012, *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering, CHASE 2012 - Proceedings*; **ISBN-13:** 9781467318242; **DOI:** 10.1109/CHASE.2012.6223031; **Article number:** 6223031; **Conference:** 2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering, CHASE 2012, June 2, 2012 - June 2, 2012; **Sponsor:** CFP1221G-ART; **Publisher:** IEEE Computer Society
Author affiliation: (1) S.e.a.l.-software Architecture and Evolution Lab, Department of Informatics, University of Zurich, Switzerland
Abstract: Code reviews are an effective mechanism to improve software quality, but often fall short in the development of software. To improve the desirability and ease of code reviews, we introduce an approach that explores how multi-touch interfaces can support code reviews and can make them more collaborative. Our approach provides users with features to collaboratively find and investigate code smells, annotate source code and generate review reports using gesture recognition and a Microsoft Surface Table. In a preliminary evaluation, subjects generally liked the prototypical implementation of our approach for performing code review tasks. © 2012 IEEE. (16 refs)
Main heading: Software engineering
Controlled terms: Computer software selection and evaluation - Gesture recognition
Uncontrolled terms: Code review - Code smell - collaboration - gesture - Multi-touch - Software metrics
Classification Code: 716 Telecommunication; Radar, Radio and Television - 723 Computer Software, Data Handling and Applications - 723.1 Computer Programming
Database: Compendex
Compilation and indexing terms, Copyright 2013 Elsevier Inc.
Data Provider: Engineering Village

73. An aspect transformation approach with refactoring

Zhou, Cliaohong (1, 2); Xu, Haowen (1, 2, 3); Zhou, Tiallin (1, 2); Shi, Liang (1, 2)
Source: *17th International Conference on Software Engineering and Knowledge Engineering, SEKE 2005*, p 318-323, 2005, *17th International Conference on Software Engineering and Knowledge Engineering, SEKE 2005*; **ISBN-13:** 9781627486590; **Conference:** 17th International Conference on Software Engineering and Knowledge Engineering, SEKE 2005, July 14, 2005 - July 16, 2005; **Publisher:** Knowledge Systems Institute Graduate School
Author affiliation: (1) Department of Computer Science and Engineering, Southeast University, Nanjin 210096, China (2) Jiangshu Institute of Software Quality, Nanjin 210096, China (3) National Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China

Abstract: Aspect-oriented programming (AOP) is a revolutionary programming paradigm aims to modularize crosscutting concerns in software design and implementation phase by a special unit-aspect, and it is proposed to resolve code tangling in object-oriented programming (OOP). But the A OP languages cannot be compiled directly by current compilers, so a transformation is needed. Refactoring is the process of improving the design of existing code without altering the external behavior of the program In this paper, we first present a preprocessor based on information tables for Aspect J method-call join point, then we refactor AspectJ programs to make it weaving-friendly, at last the weaving engine transfers AspectJ into pure Java programs, which can he compiled directly bv current compilers. Our approach is based on source code, therefore it can be applied to other languages, and it can be a platform for aspect mining and AO-refactoring further. (15 refs)

Main heading: Java programming language
Controlled terms: Aspect oriented programming - Computer software - Knowledge engineering - Object oriented programming - Program compilers - Weaving
Uncontrolled terms: Aspect-J - Aspect-Oriented Programming (AOP) - Cross-cutting concerns in software - External behavior - Objectoriented programming (OOP) - Program transformations - Programming paradigms - Refactorings
Classification Code: 723 Computer Software, Data Handling and Applications - 819.5 Textile Products and Processing
Database: Compendex