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

Data Provider: Engineering Village

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

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

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

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

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

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

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

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

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

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

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

Data Provider: Engineering Village