**LITERATURE SURVEY**

**1) Feature identification: A novel approach and a case study**

**AUTHORS:**  G. Antoniol and Y.-G. Gueheneuc

Feature identification is a well-known technique to identify subsets of a program source code activated when exercising a functionality. Several approaches have been proposed to identify features. We present an approach to feature identification and comparison for large object-oriented multi-threaded programs using both static and dynamic data. We use processor emulation, knowledge filtering, and probabilistic ranking to overcome the difficulties of collecting dynamic data, i.e., imprecision and noise. We use model transformations to compare and to visualise identified features. We compare our approach with a naive approach and a concept analysis-based approach using a case study on a real-life large object-oriented multi-threaded program, Mozilla, to show the advantages of our approach. We also use the case study to compare processor emulation with statistical profiling.

**2) Feature identification: An epidemiological metaphor**

**AUTHORS:** G. Antoniol and Y.-G. Gueheneuc

Feature identification is a technique to identify the source code constructs activated when exercising one of the features of a program. We propose new statistical analyses of static and dynamic data to accurately identify features in large multithreaded object-oriented programs. We draw inspiration from epidemiology to improve previous approaches to feature identification and develop an epidemiological metaphor. We build our metaphor on our previous approach to feature identification, in which we use processor emulation, knowledge-based filtering, probabilistic ranking, and metamodeling. We carry out three case studies to assess the usefulness of our metaphor, using the "save a bookmark" feature of Web browsers as an illustration. In the first case study, we compare our approach with three previous approaches (a naive approach, a concept analysis-based approach, and our previous probabilistic approach) in identifying the feature in MOZILLA, a large, real-life, multithreaded object-oriented program. In the second case study, we compare the implementation of the feature in the FIREFOX and MOZILLA Web browsers. In the third case study, we identify the same feature in two more Web browsers, Chimera (in C) and ICEBrowser (in Java), and another feature in JHOTDRAW and XFIG, to highlight the generalizability of our metaphor

**3) Debugadvisor: A recommender system for debugging**

**AUTHORS:** B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V.

Vangala

In large software development projects, when a programmer is assigned a bug to fix, she typically spends a lot of time searching (in an ad-hoc manner) for instances from the past where similar bugs have been debugged, analyzed and resolved. Systematic search tools that allow the programmer to express the context of the current bug, and search through diverse data repositories associated with large projects can greatly improve the productivity of debugging This paper presents the design, implementation and experience from such a search tool called DebugAdvisor.

The context of a bug includes all the information a programmer has about the bug, including natural language text, textual rendering of core dumps, debugger output etc. Our key insight is to allow the programmer to collate this entire context as a query to search for related information. Thus, DebugAdvisor allows the programmer to search using a *fat query*, which could be kilobytes of structured and unstructured data describing the contextual information for the current bug. Information retrieval in the presence of fat queries and variegated data repositories, all of which contain a mix of structured and unstructured data is a challenging problem. We present novel ideas to solve this problem.

We have deployed DebugAdvisor to over 100 users inside Microsoft. In addition to standard metrics such as precision and recall, we present extensive qualitative and quantitative feedback from our users.

**4) Expectations, outcomes, and challenges of modern code review**

**AUTHORS:** A. Bacchelli and C. Bird

Code review is a common software engineering practice employed both in open source and industrial contexts. Review today is less formal and more lightweight than the code inspections performed and studied in the 70s and 80s. We empirically explore the motivations, challenges, and outcomes of tool-based code reviews. We observed, interviewed, and surveyed developers and managers and manually classified hundreds of review comments across diverse teams at Microsoft. Our study reveals that while finding defects remains the main motivation for review, reviews are less about defects than expected and instead provide additional benefits such as knowledge transfer, increased team awareness, and creation of alternative solutions to problems. Moreover, we find that code and change understanding is the key aspect of code reviewing and that developers employ a wide range of mechanisms to meet their understanding needs, most of which are not met by current tools. We provide recommendations for practitioners and researchers.

**5) Leveraging usage similarity for effective retrieval of examples in code repositories**

**AUTHORS:** S. K. Bajracharya, J. Ossher, and C. V. Lopes

Developers often learn to use APIs (Application Programming Interfaces) by looking at existing examples of API usage. Code repositories contain many instances of such usage of APIs. However, conventional information retrieval techniques fail to perform well in retrieving API usage examples from code repositories. This paper presents Structural Semantic Indexing (SSI), a technique to associate words to source code entities based on similarities of API usage. The heuristic behind this technique is that entities (classes, methods, etc.) that show similar uses of APIs are semantically related because they do similar things. We evaluate the effectiveness of SSI in code retrieval by comparing three SSI based retrieval schemes with two conventional baseline schemes. We evaluate the performance of the retrieval schemes by running a set of 20 candidate queries against a repository containing 222,397 source code entities from 346 jars belonging to the Eclipse framework. The results of the evaluation show that SSI is effective in improving the retrieval of examples in code repositories.