Platform Independent Software Development Monitoring: Design of an Architecture

Mária Bieliková, Ivan Polášek, Michal Barla, Jozef Tvarožek,
Eduard Kuric, Karol Rástočný

Institute of Informatics and Software Engineering, Faculty of Informatics
and Information Technologies, Slovak University of Technology
Ilkovičova 2, Bratislava, Slovakia
{name.surname}@stuba.sk
http://fiit.stuba.sk

Abstract. Many of software engineering tools and systems are focused to monitoring source code quality and optimizing software development. Many of them use similar source code metrics to solve different kinds of problems. This inspired us to propose an environment for platform independent code monitoring, which supports employment of multiple software development monitoring tools and sharing of information among them to reduce redundant calculations. In this paper we present design of an architecture of the environment, whose main contribution is employing (acquiring, generating and processing) information tags - descriptive metadata that indirectly refer source code artifacts, project documentations and developers activity via document models and user models. Information tags represent novel concept unifying traditional content based software metrics with recently developed activity-based metrics. We also describe prototype realization of the environment within project PerConIK (Personalized Conveying Information and Knowledge), which proves feasibility and usability of the proposed environment.

Keywords: Information tag, Source code, Software development, Developer's expertise, Interaction data, Software metrics

1 Introduction

Source code quality and optimization of software development process fall within long-term problems of software engineering. Many of proposed methods that aim to solve these problems utilize source code metrics (e.g., LLOC, CLOC) [7], watch activities of developers and process of the development [6] and visualize results in different views that simplifies identification of problematic source code and communication with stakeholders [5]. These methods have different strengths and weaknesses related to particular problems. Thus by combining several methods in one environment we can achieve a more robust solution. Even more, we can move from a separate execution of distinct methods into an "orchestration", where particular methods take advantage of and reuse shared "knowledge" base about source codes, project documentations and developers.

Current trends of information and knowledge modelling utilize ontologies for sharing and storing information [19]. Ontologies (either lightweight or heavyweight) are also often used in web information services that describe webpage artifacts, which are in some aspects similar to software source code artifacts. This description of webpage artifacts is provided via semantic annotations that can refer to concepts of external ontologies [1] or they can directly contain fragments of an ontology in form of bags of triples [17]. These semantic annotations can be put directly into web pages as HTML tag attributes, practically visible only to machines and not distracting user in any way. Such approach however cannot be used for source codes, as they would quickly become unclear, hard to read and understand for a programmer.

Another approach is to store metadata in external files with proper references to described resources. This approach suffers by granularity issues – metadata can often refer only the whole file [15], what is insufficient for source codes, or they straightly refer line numbers of annotated source code artifacts [10] what significantly decreases maintainability of metadata (after each source code file modification, all references have to be updated to current line numbers).

In this paper we introduce design of an architecture of a novel environment for code monitoring which employs *information tags* as descriptive metadata over document model and user model. This way we contribute to solving a problem of computational redundancy and increase cooperation among services and tools for software development support.

2 Architecture Overview

The proposed environment has to be able process data created by developers directly as well as indirectly (by observing and logging their actions) and to provide added value in real-time. For this reason we divide processing data to multiple partial processes that are sub-processes of two main processes – *data acquisition* and *added value provision*.

2.1 Data Acquisition

Decomposition of data processing into multiple processes gives us a possibility to break complex problems of processing big data to several smaller and less complex problems that can be distributed over multiple machines and processed in real time. This can be especially notable in the process of data acquisition, which needs to cope with stream of events about modifications in source code, projects documentations and learning materials and also activities of developers and team leaders. Thus we decompose data acquisition to four horizontal layers – *data source*, *data*, *metadata* and *metadata processing* based on granularity of the data and to three vertical layers - *tags*, *documents* and *logs* about activities based on the character of the data (see Fig. 1).

Data acquisition starts in data source – users' *working environments*. Data from working environments are collected via a set of pre-installed tools that

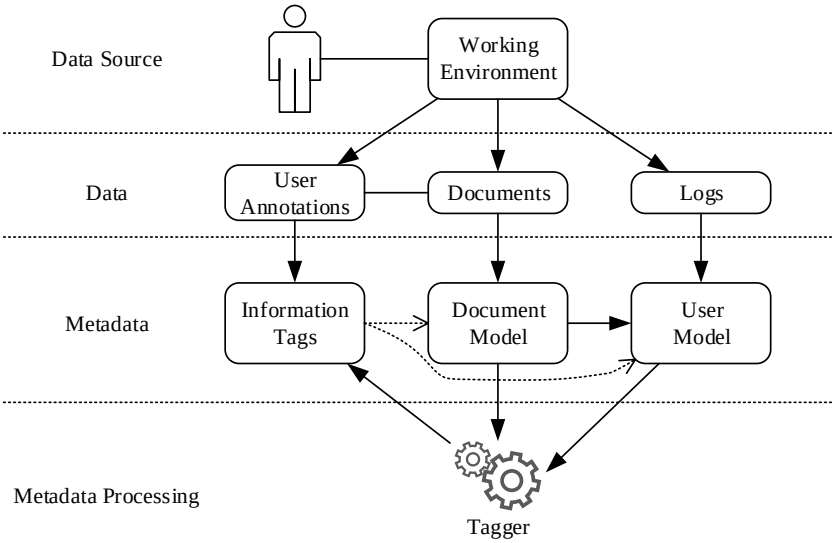

Fig. 1. Horizontal layers of data acquisition view. All elements are independent software systems with SOAP a REST-based interfaces that make stored objects accessible via URI identifiers and queries.

monitor activities of developers in their integrated development environments, web browsers and office tools and collect contextual (possibly including also biometrics) information. The collected data are stored into repositories at the data layer, from which they are processed into *document models* and *user models*. All modifications in models are streamed as events to the *tagger*, a component which processes them and creates information tags - descriptive metadata assigned to the objects from document models and user models (we discuss "information tags" and motivation for creating them in the section 3.3).

2.2 Added Value Provision

Processing the acquired data to metadata in formats of models and information tags unburdens individual methods/services from recurrent preprocessing of raw data and redundant calculations. This does not only save computation resources, but it also saves data traffic as they do not have to access whole raw data but they only query for necessary metadata. Sources of queried data depend on roles of end users – consumers of methods and services. In our environment we differentiate two main roles of users - *developers* and *team leaders* (see Fig. 2).

Developers work directly with software project documents (e.g., source code, documentations). The most of documents used by developers are stored in developer' machines in needed versions or they are synchronized with documents repositories by specialized tools (e.g., IDE, office tools). Therefore services of the environment will do redundant processing if they load and work directly

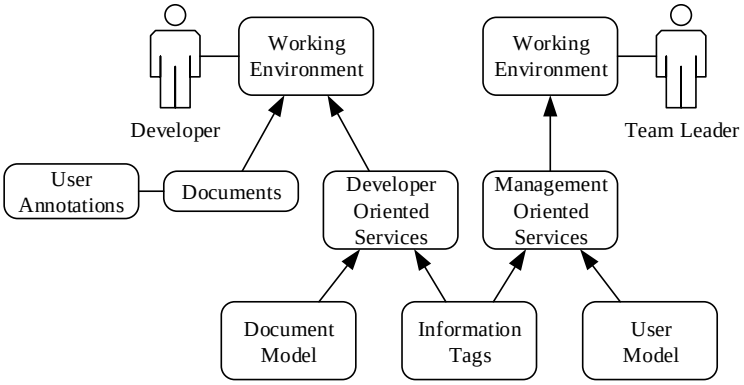

Fig. 2. Enriching users' working environment with added value provided by the code monitoring environment

with these documents. It is more efficient to perform calculations and processing over document models and information tags and then send results to developers' machines, where they can be merged back into the documents.

Management oriented services work similarly to *developer oriented services*. Both work only with metadata without necessity of an access to the whole raw data. Difference is in metadata stored in models that are processed by management oriented services. These services do not need to work with document models. Managers do not need to read documentations or source code, they need information about their teams (e.g., skills of developers). Therefore management oriented services query the user model and information tags and present results through specialized tools or web applications.

3 Vertical Layers

3.1 Documents

During software development, developers write and analyze source code, create and study different kinds of texts (e.g., specifications, API, tutorials), or use Q&A sites and community forums for finding/providing solutions (e.g., code examples). It is rich information space which includes resources in a natural language as well as a "spiderweb" of software artifacts. Whether it is a file/text in a natural language or a programming language or it is a resource located on the Web or in a local repository, everything is a document.

We classify documents into three classes: *source code*, *web pages* and *project documentation*. Although, this classification seems to be straightforward, note that there are also different relationships among the documents. For example, a code snippet can be copied from a web page into local code or documentation. The relationship information between documents (source, target) is captured in user activity logs and stored in information tags.

3.2 Logs

We have developed several agents (tools) that collect and process (existing) documents and user activities. They allow us to capture, track, analyze and evaluate different events. We focus on monitoring developers' work in IDE, their activities in a web browser and events of an operating system. To capture coding/working activities we use supporting tools/plug-ins (e.g., plug-ins for Microsoft Visual Studio IDE, Eclipse IDE and Firefox).

In an IDE we capture activities such as open/add/edit file (associated with a solution/project), check-in, debug, copy/paste, code focus and selection, built-in find, code refactoring, and stream of edits. In a web browser we capture activities such as search on the Web (keywords, target URLs), find on a page, entering URLs, manipulation with tabs, content selection, creating and using bookmarks. For monitoring other activities we use OS Monitor. It allows us to capture and monitor running applications, utilization of hardware resources, biometric information (keyboard, mouse) and performed activities in office tools (e.g., Microsoft Office Word, Microsoft Office OneNote).

Each agent collects activities, transforms them to logs and sends the logs to the Local Logging Service (LLS). The task of LLS is to gather the logs. A delivered log from an agent to LLS can be set so-called "flag milestone". It means that LLS sets up a package of the gathered logs and sends the package to the Server Logging Service (SLS) that stores the logs into a database (see Fig. 3).

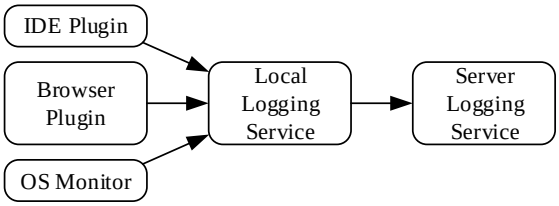

Fig. 3. Dataflow of collected logs

3.3 Information Tags

Information tag is a descriptive metadata with a semantic relation to a tagged content. It is an extension to the common concept of a tag as a simple keyword or phrase assigned to an artifact. Information tag adds additional value (semantics) to the software artifact itself. For example, information tags can be product of monitoring signals of explicit and implicit feedback generated by developers working on source code. Information tag is defined by a triple of [11]:

- *Type* - defines a type and a meaning of the information tag;
- *Anchoring* - identifies a tagged information artifact;
- *Body* - represents a structured information, a structure of which corresponds to the type of the information tag.

We distinguish information tags according their source: user created and machine generated information tags. *User created information tags* are created by users via specialized tools integrated into their working environment (e.g., in a form of a plug-in for IDE). They are directly readable for users. This makes user created information tags easily understandable and usable, what gives the environment advantage of naturally collecting users' knowledge about tagged objects (e.g., ratings of classes). In this manner user created information tags generally have a clear meaning for our users (e.g., provide review feedback attached to a source code). They can also be utilized to train or evaluate methods for enhancement of software development (e.g., automatic identification of unreliable or risky source code).

Machine generated information tags are an analogy of user created information tags for programme services. It is a tag which contains structured machine-readable information which has a meaning with its interconnection with a tagged content and/or its context (e.g., environment, history). For example, if the information tag "Edited 23 times" is defined, its information has meaning for us only if we look up at tagged method in a source code file, which has been edited. This way information tags represent form of lightweight semantics, in which any service can store and share its information related to objects (or any part of an object) of an information space (e.g., source code fragments). As a result, information tags decrease redundancy of data processing when some services need common partial results and also allow employing data mining techniques.

Not all source code annotations created by a user (*user annotations* in Fig. 1) can be considered as information tags, i.e. a descriptive metadata expressed in defined structure. The diversity of machine generated and user created information tags is in the logical level, in which user created information tags are always metadata straightly understandable to users. In the realization of the environment (see following section) we implement common model, repository and maintaining services for user created and machine generated information tags.

4 Case Study: Code Monitoring in Software House Environment

To evaluate feasibility of the proposed environment we have developed its prototype realization within the research project PerConIK (Personalized Conveying of Information and Knowledge). The project is focused on support of enterprise applications development in a software company by considering a software system as a web of information artifacts [3]. We experiment also with the development of students' team projects in master study programmes in Information Systems and Software Engineering at the slovak University of Technology in Bratislava. Project leader is a medium size software company so that all monitoring is realized in accordance with defined company policies.

In this section we describe core parts of the realized environment (e.g., information tags management) and several methods/services with partial results.

4.1 Infrastructure and Metadata Management

Information Tags Repository. The main innovation in architecture of the presented environment lies in employing information tags. To utilize advantages of information tags we proposed information tags repository which respects following requirements [4]:

- The repository has to be scalable - it has to have good read and modify performance despite of nontrivial number of stored information tags.
- The repository has to be able to store information tags in a freeform model which can be easily extended with new information tag types.
- Due to semantic meaning of information tags, inference has to be supported.

To fulfill these requirements we combine advantages of RDF and document stores. RDF has advantage in possibility of freeform data modeling and inference possibilities but it is at the expense of time complexity in the case when whole information has to be loaded (multiple SPARQL queries have to be processed while each query can take several seconds [14]). On the other side, document stores have good access to whole objects but they do not support inference [18].

Our repository is based on MongoDB[1] database which stores information tags in the object model based on Open Annotation Data Model[2]. We utilized standardized Open Annotation Data Model for its prevalence in annotation systems and straight analogy between information tags and annotations (both have a type, a body and an anchoring). We solved problem of missing support for inference in MongoDB by proposition of MapReduce-based SPARQL query processing algorithm [4]. We also performed several usability evaluations with our prototype realization. Executed test cases proved that proposed information tag repository provide enough performance for use cases of the environment and also that proposed SPARQL query processing algorithm reached almost same time as processing SPARQL queries as native horizontally scalable RDF storage BigData[3]. Our results are in detail described in [4].

Information Tags Maintenance. Information tags are linked with problem of their maintenance. This is especially visible in the case of information tags anchored to source code. Source code files are continuously modified, deleted and created. It leads to several problems of information tags maintenance:

- Generating missing information tags - newly written source code files or their parts have to be tagged with information tags that describe new source code.
- Repairing affected information tags - each modification in a source code file can affect validity of information tags at two levels - validity of body and anchoring of information tags. In addition, information tags' bodies can be affected by time - information that are stored in them can become obsolete.

[1] http://www.mongodb.org
[2] http://openannotation.org/spec/core/
[3] http://www.systap.com/bigdata.htm

– Removing invalid information tags - unrepairable information tags or information tags those targets are missing (have been deleted) have to be deleted from the information tags repository.

The first step of information tags maintenance is repairing invalidated information tag anchoring. This has to be done because we have to be able locate right source code artifacts where information tags have to be anchored before providing necessary maintenance of affected information tags. To solve this problem we do not employ any special process or service. We made a decision to design a robust location descriptor suitable for source code with algorithms for its building and interpreting [12]. It give us possibility to recalculate positions in real time (during editing code) without necessity to load previous versions of code.

Remaining maintenance tasks are provided by *tagger*. Tagger is a rule-based service which collects streams of events about modifications in user and document models and in case of fulfillment of a rule's condition it performs actions described in the rule. The tagger's core is based on linked stream data processing [9]. We employ C-SPARQL engine [2], which processes RDF graphs of events from models updating services. Employment of linked stream data [16] increases inference possibilities over events and decreases memory complexity of rule execution (incremental events processing). In addition tagger uses inferred results of fulfilled C-SPARQL query-based conditions in simple rules' actions.

Presentation of Tags. Information tags are primarily designed for services, but some of them can have direct added value for developers too. E.g., a service, which watches developers' activity, can automatically assign information tags with authorship to source code artifacts. Such information tags can be important for a developer that has to refactor older source code.

In addition some information tags and especially user generated information tags could not be maintained automatically. In these cases developers have to manually maintain invalidated information tags. For these reasons we implemented the plug-in for Microsoft Visual Studio 2012 for visualizing information tags (see Fig 4). Small graphic symbols (on the left side) pointed to concrete tags in the source code: green triangles for single tags and two *halftriangles* with connector for the pair tags in relation. On the right-hand side of the editor we can display the labels with the content of the tag: authors of the particular source code (after the keyword *by*), users of the source code (after the keyword *used by*), ranking (green stars are positive ranking and red stars are negative), topics, patterns or antipatterns. On the bottom of the editor we built filters and also we can activate infotip on the information tag label which completes whole information (here in the figure for example only the number of commits, the *author* of the tag (generated tag by the *tagger*), creation time, etc.).

4.2 Infrastructure Usage Possibilities

The information tags allow us to design and develop wide range of methods/services focused, e.g., to automatically evaluate developers' expertise, to discover a de-

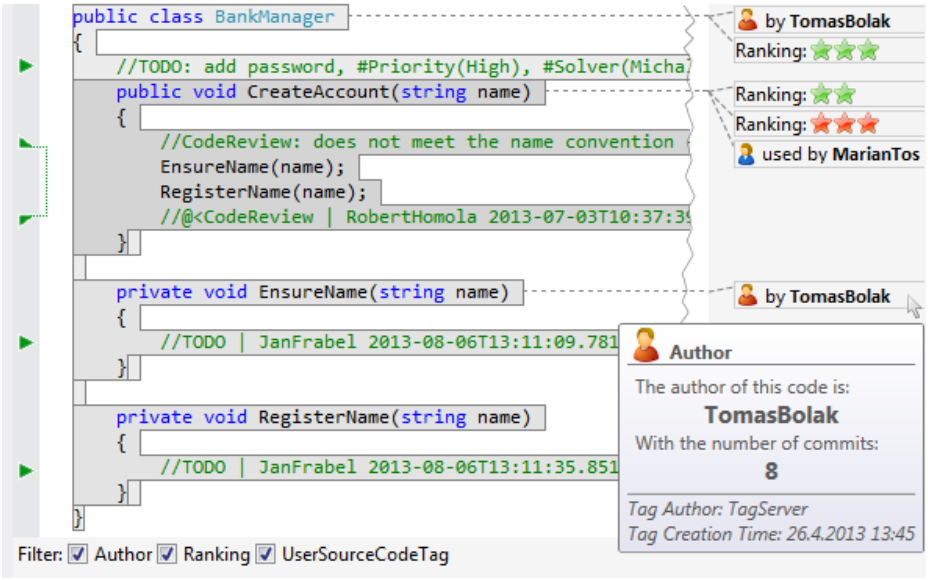

Fig. 4. Visualization of information tags in Microsoft Visual Studio 2012

gree of developers' productivity and effectiveness, to reveal developers' practices and habits or to establish quality of created code.

Modeling Developer's Karma. One possible employment of proposed infrastructure for software development monitoring is to model developer's expertise (karma). It is valuable in real (internal) environment of a software company, but also in academic environment. Determining a developer's expertise [13] in a software company allows for example managers and team leaders to look for specialists with desired abilities, form working teams or compare candidates for certain positions. In academic environment, automatic establishment of students' expertise allows a teacher to evaluate students' knowledge and know-how. Based on it, e.g., the teacher can adapt and modify his teaching practices. On the contrary of a software company, where software is created by professionals, in academic environment, students learn how to design and develop software. Therefore, the establishment of developer's expertise requires different approaches.

One possible scenario we work on is to establish automatically developer's karma based on monitoring his working activities during coding in IDE, analyzing and evaluating the (resultant) code he creates and commits to a local repository. To establish the overall developer's karma for a software project we investigate information tags on software artifacts (components), which the developer creates. We take into account developer's "karma elements" as:

– *degree of authorship* – the developer's code contributions and the way how the contributions were created to a component;

- *authorship duration and stability* – the developer's know-how persistency about a component;
- *technological know-how* – the level of how the developer knows the used technologies (libraries), i.e., broadly/effectively, this includes also estimating quality of developed source code;
- *degree of productiveness* - a degree of difference between the real generated and finally used code lines in a component;
- *component importance* - a degree of importance of a component in the software project.

We established these particular developer's karma elements (metrics) based on our observation of developers' activities (logs). The overall developer's karma is calculated as a linear combination of these karma elements. Each karma element is calculated based on information tags generated while the developer works on source code or by off-line analyses of source code (those indicating quality of code developed by the developer). By applying the metrics we are able to observe and evaluate different indicators. For example, we can sight the developer who often copies and pastes source code from an external source (Web). Contributions of such developer can be relative to the software project, moreover, it can reveal a reason of his frequent mistakes or low productivity in comparison with other developers.

Search in Source Code Based on Reputation Ranking. Code search engines help developers to find and reuse software components. To support search-driven development it is not sufficient to implement a "mere" full text search over a base of code, human factors have to be taken into account as well. Reputation ranking can be a plausible way to rank code results using social factors. Trustability of code (developer's/author's reputability) is a big issue for reusing code (software components). When a developer reuses code from an external source he has to trust the work of external developers that are unknown to him. It can be supported by using an externalized model of each developer's expertise of a particular code (software component).

In search-driven development we apply our model and approach for automatic establishing developer's karma. It allows developers to rank code results not only based on relevance but also authors' reputation of the results. We exploited our know-how in implementing a search engine which in addition to relevance of code (software component) establishes its importance [8].

5 Conclusions and Future Work

We have introduced an approach to code monitoring in software projects based on information tags as descriptive metadata that provide a unifying element for reasoning on source code and developer activities represented by document and user models.

Information tags provide a basis for reasoning useful information to developers and managers similarly as metadata do for the applications on the Web. Examples are identification of bad practices, evaluation of source code quality based on an estimation of the current user state followed his activity, recommendation of good programming practices and tricks/ snippets used by colleagues. They also serve as an input for reasoning on properties of software artifacts such as similarity with code smells, estimation of developer skill and proficiency [3]. Information tags are stored in the information tags repository which is designed with great emphasis on scalability and the ability to store information tags in a freeform model which can be easily extended with new information tag types. Information tags are linked with a problem of their maintenance. We have introduced solutions for repairing and removing invalid information tags.

Our approach allows to design and develop wide range of methods/services, e.g. automatic evaluation of developers' expertise or establishment of quality of created code. Although, in this paper we present an approach for modeling developer's karma, in our research we also use and apply the proposed approach in modeling developer's emotion and investigation of the influence of the detected emotion on the quality of created code and recommendation of software artifacts to a developer during working in IDE.

In future work, our primary goal is to finish the implementation of our core services and to perform their final evaluation. Next we plan to deploy the implemented prototype in a software company and at the University in the subject called "Team project" where students develop relatively large software systems. We also plan to propose and realize additional supporting services, e.g., a service for establishment of code quality based on context - i.e. developer‘s position (work, home) or weather. Our final aim is to improve development efficiency and software quality during its evolution.

Acknowledgments. This contribution/publication is the partial result of the Research & Development Operational Programme for the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

1. Araujo, S., Houben, G.J., Schwabe, D.: Linkator: Enriching web pages by automatically adding dereferenceable semantic annotations. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010, LNCS, vol. 6189, pp. 355–369. Springer-Verlag, Berlin, Heidelberg (2010)
2. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for c-sparql queries. In: Proc. of the 13th Int. Conf. on Extending Database Tech. pp. 441–452. ACM, New York (2010)
3. Bieliková, M., Návrat, P., Chudá, D., Polášek, I., Barla, M., Tvarožek, J., Tvarožek, M.: Webification of software development: General outline and the case of enterprise application development. In: Proc. of 3rd World Conf. on Inf. Tech. (WCIT-2012). pp. 1157–1162. WCIT'12, University of Barcelon, Barcelona (2013)

4. Bieliková, M., Rástočný, K.: Lightweight semantics over web information systems content employing knowledge tags. In: Castano, S., Vassiliadis, P., Lakshmanan, L., Lee, M. (eds.) ER 2012, LNCS, vol. 7518, pp. 327–336. Springer-Verlag, Berlin, Heidelberg (2012)
5. Bohnet, J., Döllner, J.: Monitoring code quality and development activity by software maps. In: Proc. of the 2nd Workshop on Managing Technical Debt. pp. 9–16. ACM, New York (2011)
6. Fritz, T., Murphy, G.C., Hill, E.: Does a programmer's activity indicate knowledge of code? In: Proc. of the the 6th Joint Meeting of the European Soft. Eng. Conf. and the ACM SIGSOFT Symposium on The Foundations of Soft. Eng. pp. 341–350. ACM, New York (2007)
7. Kothapalli, C., Ganesh, S.G., Singh, H.K., Radhika, D.V., Rajaram, T., Ravikanth, K., Gupta, S., Rao, K.: Continual monitoring of code quality. In: Proc. of the 4th India Software Eng. Conf. pp. 175–184. ACM, New York (2011)
8. Kuric, E., Bieliková, M.: Search in source code based on identifying popular fragments. In: Emde Boas, P., Groen, F., Italiano, G., Nawrocki, J., Sack, H. (eds.) SOFSEM 2013, LNCS, vol. 7741, pp. 408–419. Springer-Verlag, Berlin, Heidelberg (2013)
9. Le-Phuoc, D., Xavier Parreira, J., Hauswirth, M.: Linked stream data processing. In: Eiter, T., Krennwallner, T. (eds.) Reasoning Web. Semantic Technologies for Advanced Query Answering, LNCS, vol. 7487, pp. 245–289. Springer-Verlag, Berlin, Heidelberg (2012)
10. Priest, R., Plimmer, B.: Rca: experiences with an ide annotation tool. In: Proc. of the 7th ACM SIGCHI New Zealand Chapter's Int. Conf. on HCI: Design Centered HCI. pp. 53–60. ACM, New York (2006)
11. Rástočný, K., Bieliková, M.: Maintenance of human and machine metadata over the web content. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE 2012, LNCS, vol. 7703, pp. 216–220. Springer-Verlag, Berlin, Heidelberg (2012)
12. Rástočný, K., Bieliková, M.: Metadata anchoring for source code: Robust location descriptor definition, building and interpreting. In: Decker, H., Lhotska, L., Link, S. (eds.) DEXA 2013, LNCS, vol. 8056, pp. 372–379. Springer-Verlag, Berlin, Heidelberg (2013)
13. Robbes, R., Röthlisberger, D.: Using developer interaction data to compare expertise metrics. In: Proc. of the 10th Working Conf. on Mining Soft. Repositories. pp. 297–300. IEEE Press, Piscataway (2013)
14. Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triple-store technologies for large data stores. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2007, LNCS, vol. 4806, pp. 1105–1114. Springer-Verlag, Berlin, Heidelberg (2007)
15. Schandl, B., King, R.: The semdav project: metadata management for unstructured content. In: Proc. of the 1st Int. Workshop on Contextualized Attention Metadata: Collecting, Managing and Exploiting of Rich Usage Inf. pp. 27–32. ACM, New York (2006)
16. Sequeda, J.F., Corcho, O.: Linked stream data: A position paper. In: Proc. of the 2nd Int. Workshop on Sem. Sensor Net., SSN 09. CEUR-WS, Washington (2009)
17. Tallis, M.: Semantic word processing for content authors. In: Proc. of the 2nd Int. Conf. on Knowledge Capture. Sanibel (2003)
18. Tiwari, S.: Professional NoSQL. John Wiley & Sons, Inc., Indianapolis (2011)
19. Woitsch, R., Hrgovcic, V.: Modeling knowledge: an open models approach. In: Proc. of the 11th Int. Conf. on Knowledge Management and Knowledge Tech. pp. 20:1–20:8. ACM, New York (2011)