4-2010

The implications of how we tag software artifacts: Exploring different schemata and metadata for tags

Christoph TREUDE
Singapore Management University, ctreude@smu.edu.sg

Margaret-Anne STOREY

Citation

TREUDE, Christoph and STOREY, Margaret-Anne. The implications of how we tag software artifacts: Exploring different schemata and metadata for tags. (2010). *Web2SE '10: Proceedings of the 1st Workshop on Web 2.0 for Software Engineering, Cape Town, South Africa, 2010 May 4*. 12-13.
Available at: https://ink.library.smu.edu.sg/sis_research/8916

The Implications of How We Tag Software Artifacts: Exploring Different Schemata and Metadata for Tags

Christoph Treude, Margaret-Anne Storey
Dept. of Computer Science, University of Victoria
ctreude@uvic.ca, mstorey@uvic.ca

ABSTRACT

Social tagging has been adopted by software developers in various contexts from source code to work items and build definitions. While the success of tagging is usually attributed to the simplicity of tags, the implementation details of tagging systems vary significantly in terms of metadata, schemata and semantics. In this position paper, we argue that academia and industry should be aware of these differences and that we should start to examine their implications.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—*Integrated environments*

General Terms

Human Factors, Management

Keywords

Tagging, Schemata, Metadata

1. INTRODUCTION AND MOTIVATION

Social tagging has proven to be successful in many different areas from websites such as CiteULike[1] or flickr[2] to recent integrated development environments (IDEs) for software developers. A key success factor for tags is their simplicity: tagging is a very easy way of organizing artifacts such as work items or source code, and it does not come with administrative overhead. Still, the details of how tags are realized in different systems vary considerably. Early tagging systems such as ICICLE [1] were based on a limited, controlled vocabulary while recent systems such as IBM's Jazz[3] allow users to define their own keywords. The amount of metadata stored with a tag varies from no data at all to a complete data set of tag author, time and context. In some systems, tags can be hierarchical, and in other systems, the same tag can be applied to artifacts of different types.

While there has been some research on the use of tags in software development (e.g., [6, 7]), the effectiveness of different semantics, schemata and metadata for tags is not yet well understood. In this position paper, we examine the different ways in which tagging systems have been implemented in the context of software development, and we argue that the details of the implementation play a key role in their success. Moreover, we believe that simplicity is the key characteristic that can make tagging systems succeed or fail. We investigate how simplicity can be achieved through different characteristics.

2. DIMENSIONS OF TAGGING SYSTEMS

The idea of analyzing different dimensions of tagging systems is not new. A very detailed taxonomy is given by Marlow *et al.* [2]. They identify the following seven dimensions in the design of a tagging system:

- *Tagging rights:* Users can tag everybody's resources vs. users can only tag their own resources.
- *Tagging support:* Blind tagging (users cannot see each other's tags) vs. viewable tagging (users can see each other's tags) vs. suggestive tagging (the system suggests tags to users).
- *Aggregation:* Bag model (allows duplicate tags per resource) vs. set model (no duplicates).
- *Type of object:* Type of the resource to be tagged.
- *Source of material:* Resource is supplied by the systems vs. resource is supplied by the users.
- *Resource connectivity:* Linked vs. grouped vs. none (possible connections between the resources).
- *Social connectivity:* Linked vs. grouped vs. none (possible connections between the users).

While these dimensions apply to tagging systems used by software developers, studying tagging systems used by software developers such as ICICLE [1], TagSEA [6], IBM's Jazz, BITKit [3], Google Code[4], ConcernMapper [5] and Concern Graphs [4] reveals additional dimensions on top of Marlow's taxonomy.

[1] `http://www.citeulike.org/`
[2] `http://www.flickr.com/`
[3] `https://jazz.net/`

[4] `http://code.google.com/`

3. DIMENSIONS OF TAGGING SYSTEMS IN SOFTWARE DEVELOPMENT

In this section, we identify and discuss additional dimensions of tagging systems in software development.

Pre-defined vs. user-defined.

Most current tagging systems are based on the concept of tags as *"freely-chosen keywords or terms that are associated with or assigned to a piece of information"* [7]. However, in older tagging systems such as ICICLE [1], possible keywords were pre-defined, and software developers were not able to add new keywords to the system. In a dynamic environment such as software development, the just-in-time addition of new tags is the more promising approach.

Metadata.

Different tagging systems store different amounts of metadata. For example, in the case of tagging work items in IBM's Jazz, information such as the tag author and the time a tag was applied to a work item can only be identified by browsing the work item's history. In other systems such as TagSEA, the author and time can be explicitly added to each tag instance, and tags can be searched by their authors and creation time. In order to keep the simplicity, tag authors should not be required to add metadata. However, all metadata that can be recorded automatically should be stored to provide additional context.

Semantics.

While most tagging systems treat keywords simply as terms that are associated with artifacts, some systems go beyond that and add semantics to tags. An interesting approach is taken by labels in the issue tracker of Google Code[5], which goes beyond basic labels to support key-value labels. Key-value labels contain one or more dashes, and the part before the first dash is considered to be a field name while the part after that dash is considered to be the value. Studies will need to be conducted to understand how different semantics affect software development processes.

Hierarchies.

Some tagging systems explicitly support tag hierarchies, using a dot-notation (e.g., [6]). Keywords that have dots in them can be treated as hierarchical, and they can be displayed in tree-views. In other systems such as IBM's Jazz, some developers use the dot-notation even though there is no explicit support for hierarchies. A flexible approach that offers additional views when needed is promising.

Single type of resource vs. multiple types.

Software developers handle many different kinds of artifacts from source code and work items to build scripts. Nevertheless, many tagging systems for software developers only support tagging a single kind of artifact. One exception is TagSEA. It allows software developers to tag locations in source code – called waypoints – and artifacts such as files, and it shows different kinds of artifacts in a single view. This allows for grouping and relating different kinds of artifacts while keeping the simplicity of tags.

[5]`http://code.google.com/p/support/wiki/IssueTracker#Labels`

Integration.

Another dimension is the extent to which the tagging mechanism is integrated with other tooling. Some systems support social tagging of source code, but require the user to post code fragments on public servers before tags can be applied to code fragments (e.g., DZone Snippets[6] and ByteMycode[7]). In other systems such as IBM's Jazz or TagSEA, the tagging mechanism is part of the IDE. With the recent trend of moving the IDE into the browser, tagging artifacts online is a promising approach.

4. CONCLUSION

While social tagging has been adopted by software developers for organizing artifacts such as source code and work items, the details of the implementation vary. To understand if and how tagging can support software developers, we need to examine the differences such as varying amounts of metadata and semantics. We have identified several dimensions along which tagging systems in software development can be classified. Future work lies in investigating the advantages and shortcomings of different implementations.

5. REFERENCES

[1] L. Brothers, V. Sembugamoorthy, and M. Muller. ICICLE: groupware for code inspection. In *CSCW '90: Proc. of the Conf. on Computer-supported cooperative work*, pages 169–181, New York, 1990. ACM.

[2] C. Marlow, M. Naaman, D. Boyd, and M. Davis. Ht06, tagging paper, taxonomy, flickr, academic article, to read. In *HYPERTEXT '06: Proc. of the 17th Conf. on Hypertext and hypermedia*, pages 31–40, New York, 2006. ACM.

[3] H. Ossher, D. Amid, A. Anaby-Tavor, R. Bellamy, M. Callery, M. Desmond, J. De Vries, A. Fisher, S. Krasikov, I. Simmonds, and C. Swart. Using tagging to identify and organize concerns during pre-requirements analysis. In *EA '09: Proc. of the ICSE Workshop on Aspect-Oriented Requirements Engineering and Architecture Design*, pages 25–30, Washington, DC, 2009. IEEE.

[4] M. P. Robillard and G. C. Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *ICSE '02: Proc. of the 24th Intl. Conf. on Software Engineering*, pages 406–416, New York, 2002. ACM.

[5] M. P. Robillard and F. Weigand-Warr. Concernmapper: simple view-based separation of scattered concerns. In *eclipse '05: Proc. of the OOPSLA workshop on Eclipse technology eXchange*, pages 65–69, New York, 2005. ACM.

[6] M.-A. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, and M. Muller. How software developers use tagging to support reminding and refinding. *IEEE Trans. on Software Engineering*, 35(4):470–483, 2009.

[7] C. Treude and M.-A. Storey. How tagging helps bridge the gap between social and technical aspects in software development. In *ICSE '09: Proc. of the 31st Intl. Conf. on Software Engineering*, pages 12–22, Washington, DC, 2009. IEEE.

[6]`http://snippets.dzone.com/`
[7]`http://bytemycode.com/`