Visualization of API Experience
(Extended Abstract)

Hakan Aksu Ralf Lämmel Wojciech Kwasnik
Software Languages Team, University of Koblenz-Landau

Abstract

Developers differ in terms of the APIs that they are experienced in. Understanding such differences helps in balancing team structures and assigning developers to pending issues or hiring developers. In this extended abstract, we demonstrate how simple API-related, per-developer metrics can be visualized to give a quick overview on the API experiences of developers. Data extraction is based on mining the commit history in projects in terms of references to API elements (methods and types). We visualize experiences as maps and aster plots.

Keywords: Program Comprehension, Software Visualization, Mining Software Repositories, Developer Experience.

1 Underlying concepts

We use the term API as follows: an API is a set of types (classes, interfaces, etc.) referable by name and distributed together for use in software projects. APIs can be described by their package names, package prefixes, and types within packages. An example would be the package prefix `javax.swing` referring to Java's API for GUI programming.

Following Roover et al. [2], we use the term *domain* for an extra level of abstraction for grouping APIs. For instance, Java's APIs `java.awt` and `javax.swing` APIs contribute to the *GUI* domain. These are other examples of domains: XML (programming), Meta (programming), IO, Web (programming). A domain is a (named) set of APIs.

Our use of the term 'experience' is informed by the definition in the Oxford Dictionary, 2015: "The knowledge or skill acquired by a period of practical experience of something, especially that gained in a particular profession". In our context, the period of practical experience is the contribution to a given project in terms of commits, which are sets of changes which may involve usage of API elements (classifiers, methods, enums). Following Mockus et al. [1], we assume a suitable notion of 'experience atoms'—here related to APIs. An atom is either a *reference* to an API element or a *distinct API element* being referenced in lines changed by a commit. Thus, we use these metrics: *API references: AR*, *API elements: AE*, *Domain references: DR*, and *Domain elements: DE*.

The extraction of referenced API elements relies on a lexical approach. The extractor iterates over changed lines from the commits and tokenizes them into Java lexical units. Those lexemes or Java identifiers are considered candidates for names of API elements. These candidates are intersected with the API elements of packages imported in a changed file. The resulting set is considered a referenced API element. The examples of this extended abstract are concerned with the popular open source *libGDX* and its rich commit history.

2 Visualization as map

Fig. 1 shows a map for developer experience. The complete rectangle proxies for the references to all APIs made by all developers. The contained rectangles group experience atoms by domain. The yet further nested rectangles group experience atoms by API and developer; one color is used per API regardless of different developers. Developers are shown here per number code 1, ..., 8 for the top-8 committers and 'R' for the rest. In this manner, we can assess the contribution of each domain to the project, the contribution of each API to the project, and the significance of experience of individual developers for given APIs and domains.

3 Visualization as aster plot

We use an aster plot per-developer for a given measure such as *DR* and we show these aster plots for all (selected) developers next to each other for comparison. An aster plot is an extended pie chart (or circle chart). In addition to the angles of segments, the aster plot has different radii for each segment and a value in the center. The angle of each segment is proportional to the quantity of the relevant API metric for the given developer. The radius is proportional to the given developer's percentage of the metric for the developer with the highest value. The value in the middle is the mean of all radii on a $0 \ldots 100$ scale.

The aster plots of Fig. 2 visualize domain references for the top-8 developers. (In fact, we focus on GitHub committers rather than GitHub authors here.) Each plot shows the top-8 domains for the developer at hand where the remaining domains are summarized in *Other*. The favorite domains of each developer can

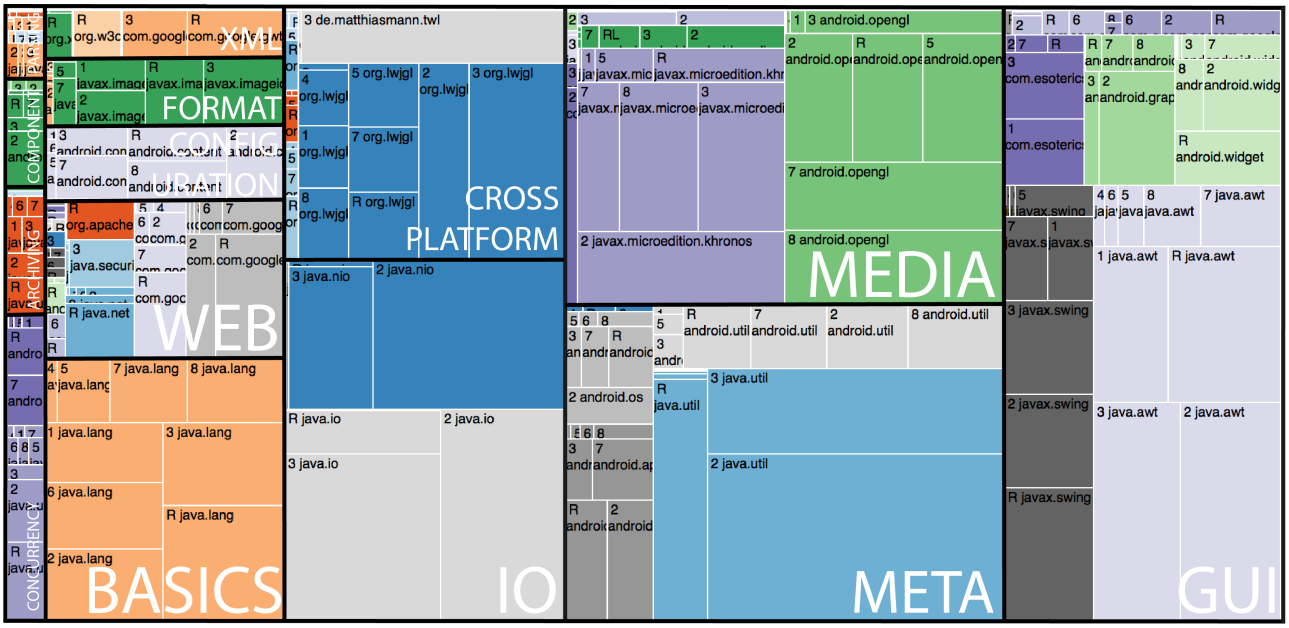

Figure 1: Domain & API references (*DR* and *AR*) for *libGDX*

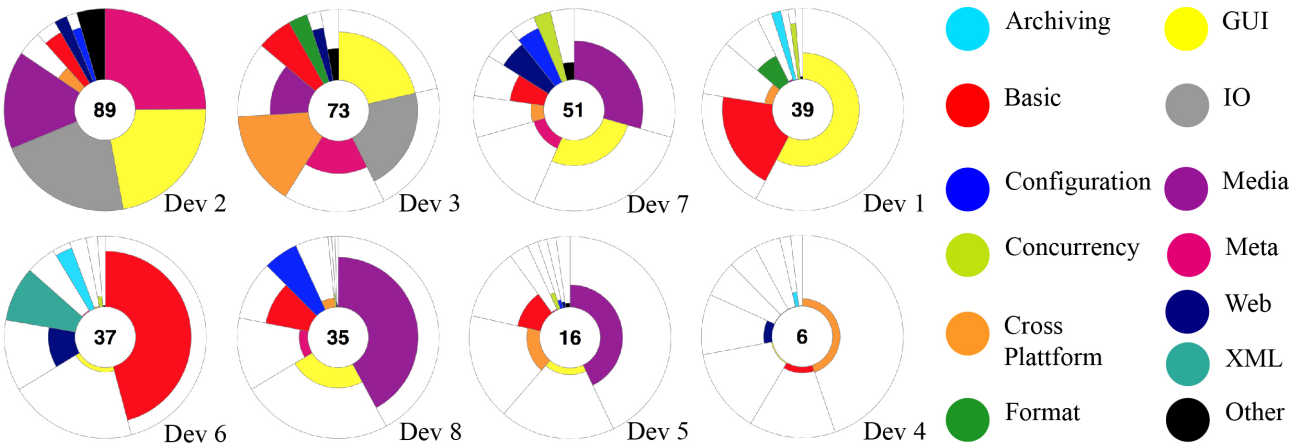

Figure 2: Per-developer comparison of domain references for *libGDX*

be easily observed in this manner. For instance, the favorite domains (with the most references) of developer 2 are *Meta*, *GUI*, *IO*, and *Media*. The developers can be easily compared by means of the radii. For instance, developer 1 is the most experienced developer in the *Archiving* domain even though it makes only a small contribution to all the experience atoms of this developer.

4 Conclusion

The developed visualization approach summarizes several aspects of per-developer API usage in a concise and systematic manner. The overall idea of interpreting API usage (in terms of both references to API elements and referenced API elements) is not new [3]; the original insight of the present work is that ratios of APIs are compared for each individual developer and across all developers while adding the abstraction level of domains on top of APIs.

In future work, we plan to develop a comprehensive visualization tool that allows one to explore per-developer API usage along different dimensions based on the selection of a collection of developers, a window on the commit timeline, a collection of APIs or domains, and yet other parameters.

References

[1] Audris Mockus and James D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proc. of ICSE 2002*, pages 503–512. IEEE, 2002.

[2] Coen De Roover, Ralf Lämmel, and Ekaterina Pek. Multi-dimensional exploration of API usage. In *Proc. of ICPC 2013*, pages 152–161. IEEE, 2013.

[3] David Schuler and Thomas Zimmermann. Mining usage expertise from version archives. In *Proc. MSR 2008*, pages 121–124. ACM, 2008.