October 2021

Identifying Software Projects That Have Matching Static or Dynamic Characteristics

Andy Lavery

Eli Daiches

Identifying Software Projects That Have Matching Static or Dynamic Characteristics

ABSTRACT

This disclosure describes techniques to analyze collections of software projects using static and dynamic features. Software engineers or operators can query the outcomes of these analyses to find cohorts of design patterns that are relevant for the task at hand in their own projects, thus helping them identify applications that meet the criteria of interest. The techniques can also be applied to generate alerts based on evolution of the matching projects, suggest code completion, and to recommend relevant example code or learning resources. Analysis of code can be performed internally on codebases controlled by a single business and/or on permitted code in a public software ecosystem. Implementation of the techniques described in this disclosure can help software engineers be guided by existing projects and potentially foster collaborations by connecting those working on similar problems.

KEYWORDS

- Software engineering
- Software development
- Static analysis
- Design pattern
- Example code
- Code completion
- Source code repository
- Code reuse
- Software project cohort

BACKGROUND

Software engineers often consult examples or recipes from other projects for application within their own projects. Consulting examples or recipes from existing code from other projects allows developers to apply past software engineering knowledge and decisions captured in the solution without starting from scratch and duplicating prior effort. Applying the insight and/or code from existing repositories can increase the velocity and quality of software development outcomes.

Although software engineers can benefit greatly from finding existing projects similar to their own, it can be challenging to identify suitable projects from among the codebases that an engineer can access. Code from projects that are dissimilar is of little use since these are typically unlikely to be applicable to the task at hand.

Existing projects that are similar to an engineer’s own project can be determined based on static as well as dynamic information about a project. Static information includes the source code and the associated libraries, components, stacks, application programming interfaces (APIs), etc. Dynamic information is based on runtime and usage parameters, such as load, performance, deployment environment, user characteristics, etc., that characterize the operation of the project under real-world conditions. Moreover, the project can be impacted by changing trends in the relevant software ecosystem applicable to it.

Static information is typically determined using tools that perform static code analysis of a software project. For instance, such analysis can indicate whether a project depends on a specific version of a software library. In contrast, dynamic information about a project can be obtained via processes that examine runtime characteristics, such as load metrics, transaction per second, disk space used by a database, etc.

Static code analysis cannot be applied to find matching projects based on runtime characteristics or API usage, and processes that examine runtime characteristics are typically not designed to help software engineers find projects with similar runtime characteristics. Moreover, no single tool offers an integrated approach that combines static and runtime analysis to help software engineers find suitable projects that match their needs.

DESCRIPTION

This disclosure describes techniques to analyze collections of software projects using static and dynamic analysis. Software engineers or operators can query the outcomes of these analyses to find cohorts of design patterns that are relevant for the task at hand in their own projects, thus helping them identify applications that meet the criteria of interest. For instance, a team of engineers can search for projects that use a specific software framework, support internal and external users, and employ a given authentication technique. Similarly, developers can search for projects that use a given software component and are capable of handling more than N queries per second. The techniques further permit observing software trends over time and alerting software engineers of changes that are likely to affect their current projects.

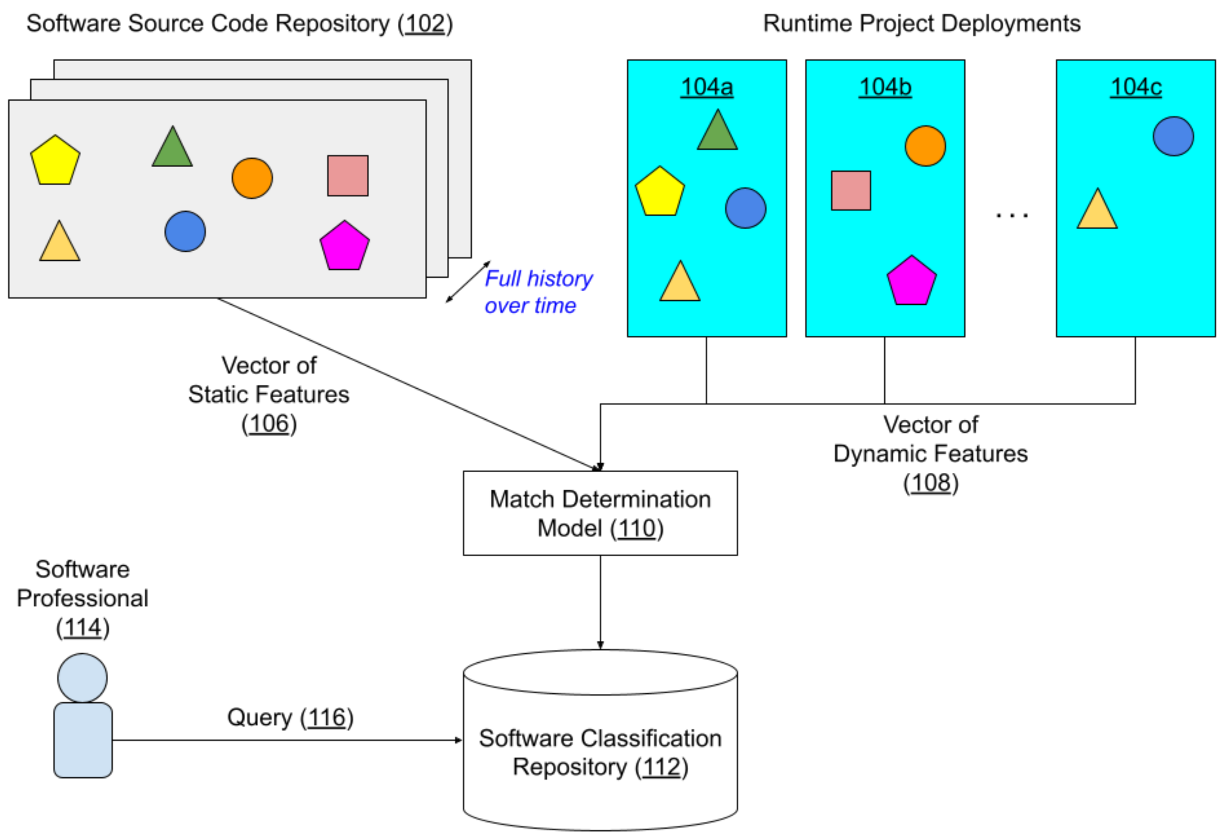
Software Source Code Repository (102)
Runtime Project Deployments
104a
104b
104c
. . .
Full history over time
Vector of Static Features (106)
Vector of Dynamic Features (108)
Match Determination Model (110)
Software Professional (114)
Query (116)
Software Classification Repository (112)

Fig. 1

Fig. 1 shows an operational implementation of the techniques described in this disclosure. Static analysis on source code history of software projects as available via a software source code repository (102) is used to generate a vector of static features of the project (106). Similarly, observation of various runtime deployments of each project (104a-c) yields a vector of its dynamic features (108). The static and dynamic features are input to a model (110) that can match the features relevant to specific queries. Since software can be updated frequently (e.g., daily, weekly, etc.), a history of the classification features can be stored in a repository (112) for determining trends over time. A software professional (114) can issue a query (116) for relevant projects to find those that match specific static or dynamic characteristics, as described in the examples above.

Static analysis can be based on various aspects of the code, such as:

- Build rules and tools;
- Static source code analysis, including specific APIs invoked, software dependencies, language(s) used;
- Deployment configuration (e.g., app manifest, cloud manifest, etc.);
- Security configuration; etc.

Observation of dynamic runtime deployments can be performed across all supported platforms and include aspects such as:

- API use based on network requests;
- API use based on runtime code coverage;
- Application traffic and performance expectations at runtime (e.g., queries per second);
- Use of specific authentication mechanisms;
- Use of specific authorization mechanisms;

- Runtime use of cloud resources;
- Aggregated user characteristics (obtained with the user’s permission), such as internal or external user, geographic information, use of human languages, device type, characteristics of the user’s network connection, etc.

The techniques described above can additionally support proactive alerts to inform software developers of relevant changes in projects similar to theirs so that they can assess whether a similar change is warranted for their own project. For instance, a software engineer might have previously queried projects matching a specific type of authentication for particular kinds of users and applied the results to the current project. Later on, if projects that match the query evolve to switch to a different authentication approach, the software engineer can be notified accordingly.

With developer permission, the techniques can be applied to enhance code completion and suggestion features within code editing applications, such as Integrated Development Environments (IDEs). For example, a developer working on configuring an authentication mechanism can be provided a recommendation of the most common code and/or settings related to that authentication mechanism as used in other relevant projects. Recommendations can be delivered using code completion features of the editor or shown separately as relevant sample code.

The recommendations can further include suggestions for learning resources or tutorials based on similarities and dependencies discovered in common patterns and stacks across cohorts of software projects. For instance, developers working on a project that shares multiple commonalities with other projects can benefit from learning about pieces in other projects that are missing in their own projects.

Apart from the source code, the techniques described above can include relevant other permitted metadata, such as design documents, configuration files, personnel lists, etc. Applying natural language processing (NLP) to such metadata can provide additional useful information for finding projects matching specific characteristics and enable support for queries using natural language. Moreover, the techniques can be leveraged when writing design documents in much the same way as their application for writing source code. For instance, consulting related design documents can help think through the design with data and ideas from similar projects.

Implementation of the techniques described in this disclosure can help software engineering projects seek guidance from other projects in the form of best practices, code examples, design patterns, etc. For instance, developers can study similar projects in terms of technology choices, runtime performance, etc. Matches obtained via the techniques can additionally help connect a development team with other teams working on similar problems and potentially foster collaboration.

The techniques can be deployed to operate internally within a single business setting. Alternatively, or in addition, the techniques can be provided in a public software ecosystem, such as that of a cloud service provider, source code hosting service, etc., with permission from users of the software platform(s). Depending on choices of developers and users, project matches can be anonymized or made visible.

Source code as well as dynamic information is accessed upon specific permission from respective project owners. Project owners can choose to deny permission, restrict permission to specific attributes or sections of code, etc. Generation of vectors of static and dynamic features is performed based only on the permitted information. Project owners are provided with options to modify permissions.

CONCLUSION

This disclosure describes techniques to analyze collections of software projects using static and dynamic features. Software engineers or operators can query the outcomes of these analyses to find cohorts of design patterns that are relevant for the task at hand in their own projects, thus helping them identify applications that meet the criteria of interest. The techniques can also be applied to generate alerts based on evolution of the matching projects, suggest code completion, and to recommend relevant example code or learning resources. Analysis of code can be performed internally on codebases controlled by a single business and/or on permitted code in a public software ecosystem. Implementation of the techniques described in this disclosure can help software engineers be guided by existing projects and potentially foster collaborations by connecting those working on similar problems.