Integrating Code Search into the Development Session

Mu-Woong Lee Seung-won Hwang
POSTECH, Korea

Sunghun Kim
HKUST, Hong Kong

Motivation

Copy-and-paste is one of the most common software development activities to support rapid development. During writing a module, reusing a code piece sharing the same syntactic structure can help developers to quickly complete the module.

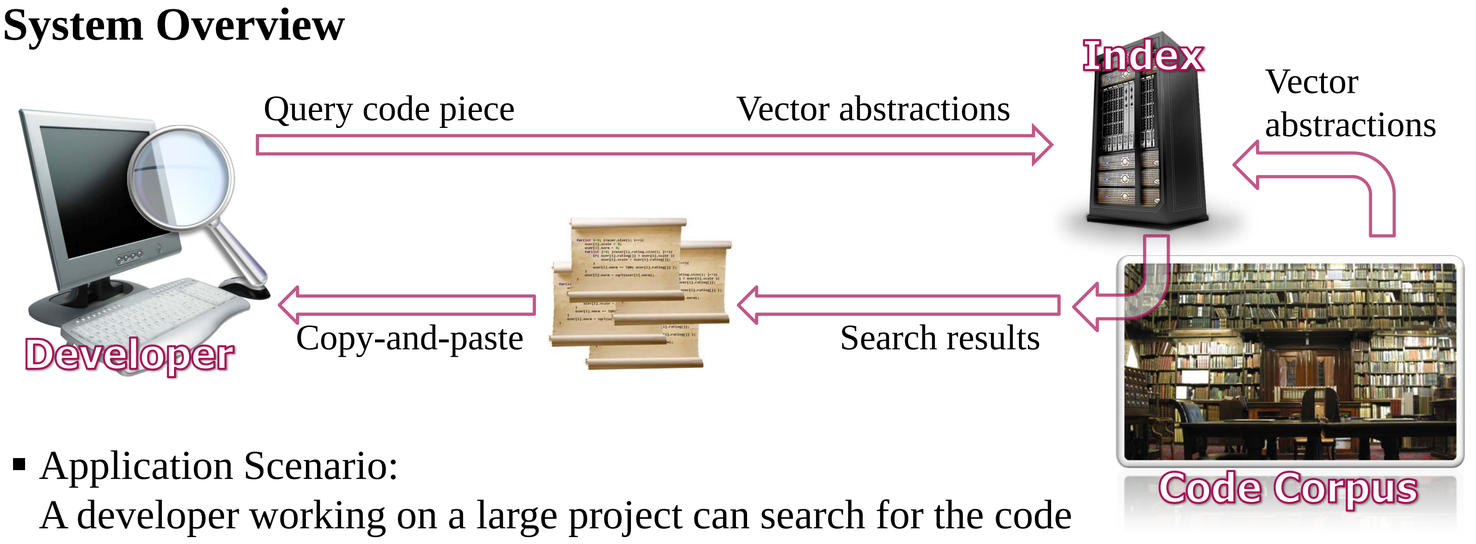
Limitations of Existing Work:

- Commercial code search engines are scalable to very large code corpus, but they disregard structural similarity.
- Code clone detection tools are often structural-aware, but they are insufficient to be "interactively" used in development sessions.

Our Goal:

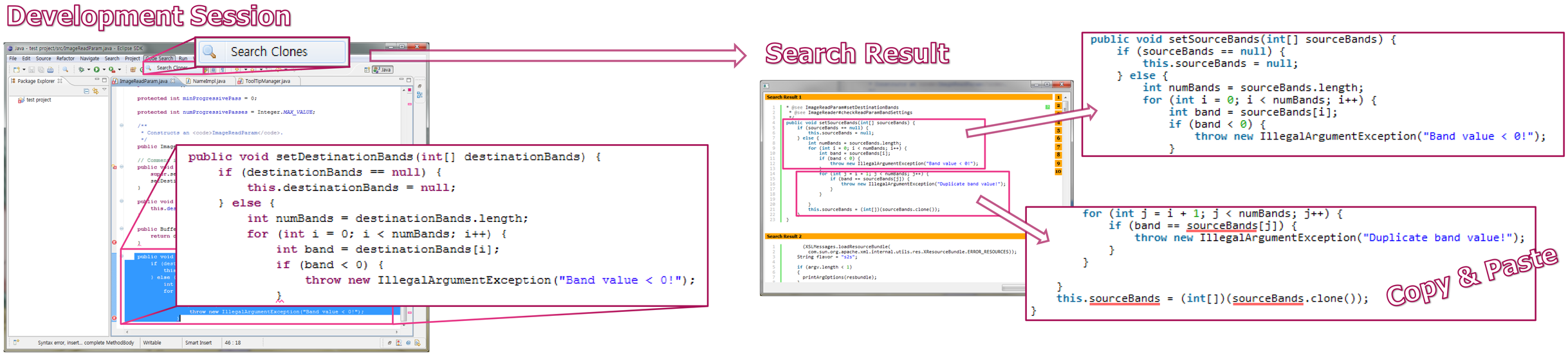
We combine the strength of the both lines of work, by enabling fast similarity searches of structurally related code.

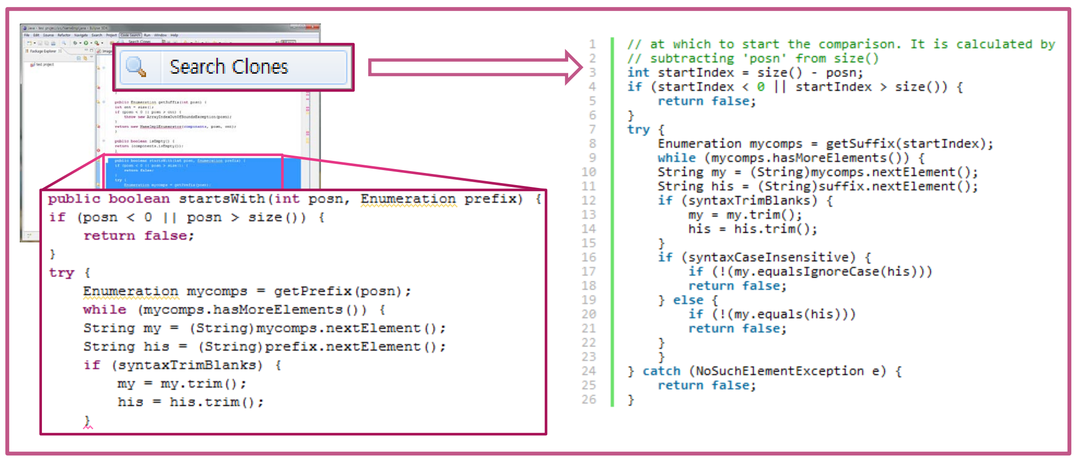
System Overview

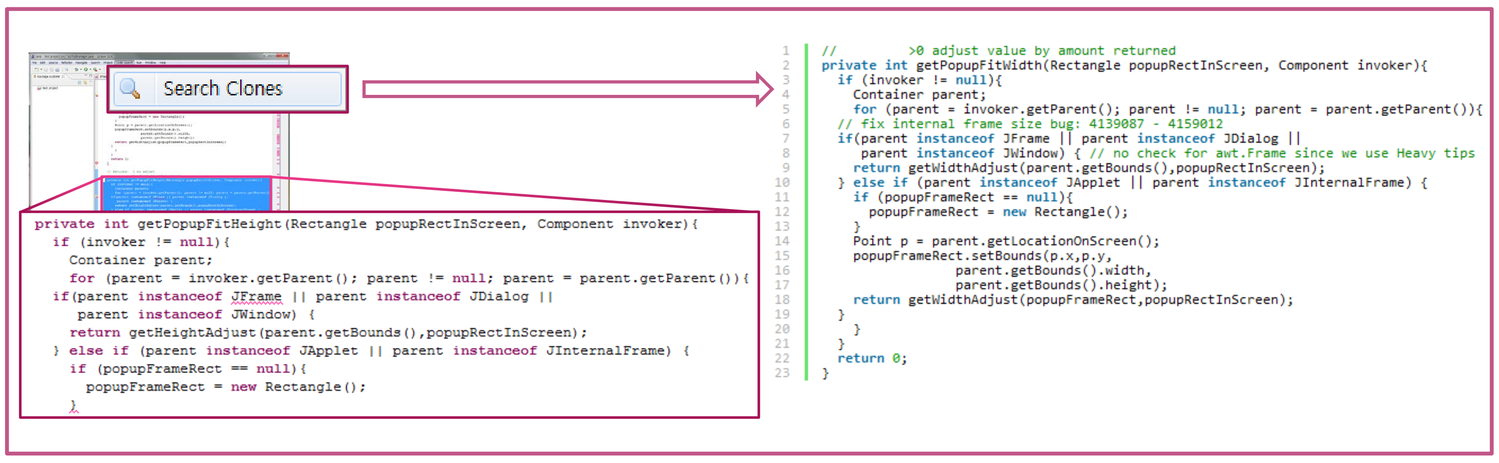


- Application Scenario:
  A developer working on a large project can search for the code sharing the same syntactic structure, then either use the results as references or copy them into his own code.
- Search Scheme:
  To support efficient structural similarity comparison between code pieces, we adopt a multi-resolution vector abstraction of code, and design index building and traversal algorithms optimized for code data and code search workload.

Demonstration







How to abstract code as vectors?

We adopt the vector abstraction used in DECKARD[1] to abstract syntactic information of code, using Abstract Syntax Tree (AST).

1) *Characteristic vectors*
   Given a code S, its AST T, if a subtree T_i of T has at least minT nodes, T_i's corresponding part S_i in S is a code piece. The characteristic vector v_i of S_i consists of occurrence counters of syntactic elements in T_i.
2) *Distances between vectors*
   The distance between two vectors is their L_2-norm.
3) *Dimensionality reduction*
   The characteristic vectors are 261 dimensional, we thus select the top-D' dimensions with the highest variances.

How to build the index?

1) *Vector packing*
   We pack vectors into blocks, where each block contains a group of nearby vectors, more precisely, raw data records of them. We then build an R*-tree of these blocks in the reduced space.
2) *Workload-aware bulkloading*
   We revised an existing bulkloading algorithm that recursively subdivides each dimension into the same number of slices. For code data, the variance of each dimension differs significantly. Our revised partitioning policy tries to partition the dataset to render more "squared" rectangles, considering the data distribution.

How to traverse the index?

Our traversal algorithm basically follows a *filtering-then-ranking* approach, but we interleaves the two steps as follows:

1) *For internal nodes*
   We traverse the index in the reduced space in a best-first manner.
2) *For leaf nodes*
   When a leaf entry is reached, we access the raw data block pointed by the entry and update the sorted list of the current known top-k clones.

For this demonstration, k is set to 20, and the search bound is 5. This means that we only report the code pieces at least 75%[1] similar to the query.

Conclusion

We demonstrated our tool enabling instant code search during development. From the demonstration, we showed how such interactive search supports rapid software development, as similarly claimed in HCI and SE communities[2], [3]. For more details about index building and traversal technology, see [4].

References

[1] L. Jiang, et al., "DECKARD: Scalable and Accurate Tree-based Detection of Code Clones," in ICSE, 2007.
[2] J. Brandt, et al., "Example-centric Programming: Integrating Web Search into the Development Environment," in SIGCHI, 2010.
[3] M. Kim, et al., "An Ethnographic Study of Copy and Paste Programming Practices in OOPL," in ISESE, 2004.
[4] **M.-W. Lee, et al., "Instant Code Clone Search," in ACM SIGSOFT/FSE, 2010.**