

Our research activity in programming languages and software engineering is directed at improving all aspects of software development, and at making software itself more efficient and reliable. Our interests span a broad spectrum and include programming paradigms and methodologies, design and implementation of programming languages, programming tools and environments, as well as programming language theory.

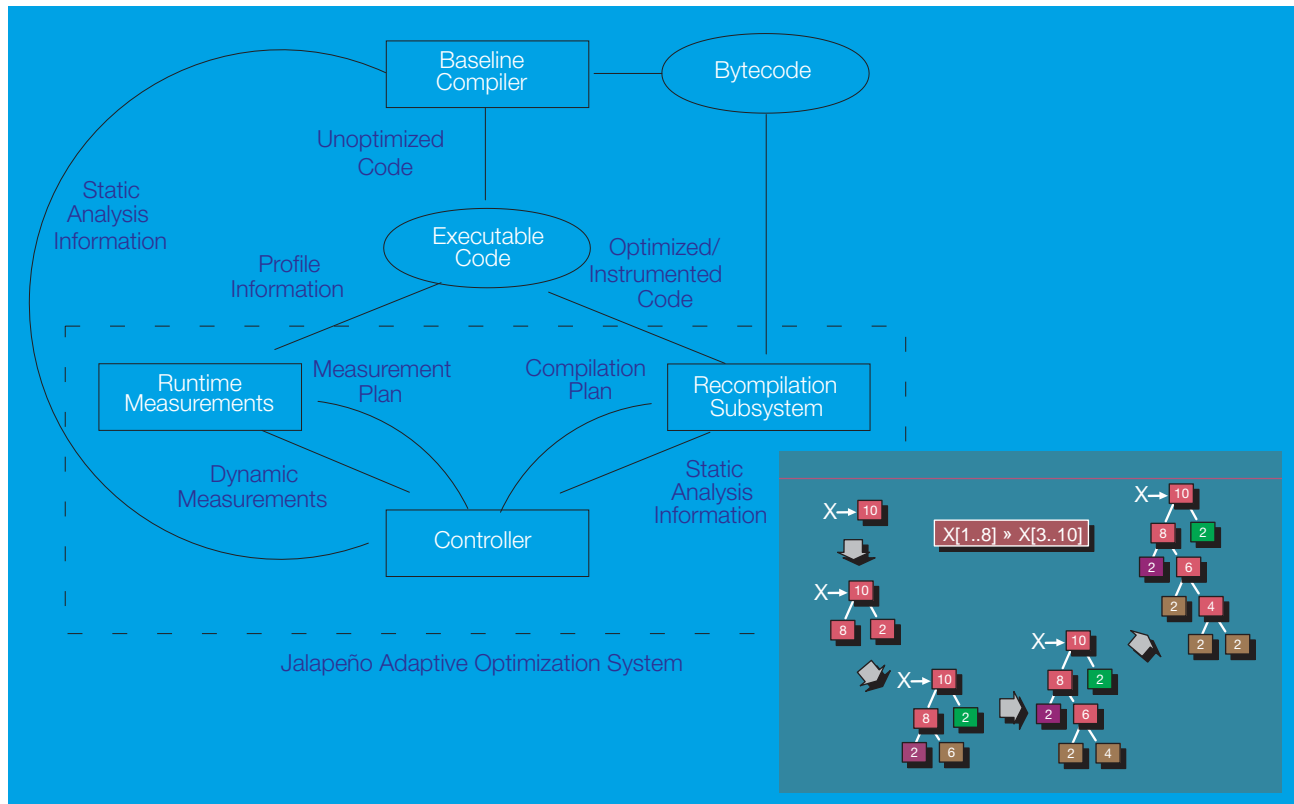
Design Patterns and Programming Paradigms

Design reuse is more valuable and more feasible than code reuse. Design patterns promote design reuse by capturing and expounding proven designs. A design pattern characterizes a single problem and its solution, the consequences of that solution, and variations on the solution's implementation. Our interests in this area include mining existing software for patterns, studying patterns application in all phases of development, and devising tools that support patterns perusal and automate their application.

Limitations in the expressive power of existing languages and programming paradigms (such as object-oriented programming) lead to software that is hard to understand, maintain, and extend. We are exploring new programming paradigms and language features that increase the expressiveness of languages. Our ongoing work on supporting multidimensional separation of concerns using hyperspaces is an example. Software modularization is highly desirable, but is usually possible along only one "dimension" (for example, by class). This constraint facilitates some development activities, but impedes others. The hyperspaces approach builds on our earlier work on subject-oriented programming to allow modularization according to multiple kinds of concerns simultaneously.

Language Design, Implementation, and Optimization

When it comes to programming languages, one size does not fit all. Domain-specific and higher-level languages simplify software development and maintenance by reducing the gap that exists between the conceptual view of a software system and its actual implementation. We are interested in language design in a number of contexts, such as rapid prototyping of business information systems, coordination of distributed systems, web-based applications, component-based programming, and concurrent programming.



New languages and language features need efficient implementations to succeed. Over the years, IBM Research has made significant contributions to this area, starting with the pioneering work of Fran Allen and John Cocke, which laid the foundations for the field of optimizing compilers. Our current interests include just-in-time compilers, mixed-mode interpreters, dynamic compilers and virtual machines. Dynamic compilation, for example, enables adaptive optimizations in which run-time information is used to choose the parts of an application that are optimized as well as in optimization of these parts. Dynamic compilation, however, also requires more efficient and innovative solutions to minimize compilation overhead. Our other interests in this area include memory management and garbage collection, efficient implementation of concurrency and synchronization, and issues in application development for embedded systems, as well as fundamental algorithms and data structures relevant to the implementation of languages and tools.

Tools and Environments

We are interested in building programming tools and environments far more powerful than the ones available today – tools that can significantly improve programmer productivity. We are interested in tools that help programmers understand, debug, and tune programs, utilizing either static information (computed via semantic analysis), or dynamic information (computed via program instrumentation), or both. Among our current interests are program visualization, component certification, tools for collaborative program development, deterministic replay of multithreaded programs, and XML processing.

Software Defect Analysis, Testing, and Verification

We are interested in methodologies, algorithms, and tools for software testing and verification. An example of our work in this area is an ongoing effort in modeling and tools related to orthogonal defect classification (ODC). ODC is a methodology for analyzing software defects, an inexpensive way to evaluate process and product during development and maintenance that can improve software quality and development efficiency. Our other interests include combinatorial optimization for test design, model-driven automatic test generation, and the use of specification and assertion languages for test automation and verification.

Jalapeño Image

