MPPR: Meaning-preserving program restructuring
Our tool allows programmers to refactor a program by applying meaning-preserving program restructurings.

It takes a program $P$ and a restructuring $R$ and produces a program $P'$ that is guaranteed to have the same input-output behavior as $P$. 
Competition

- Griswold’s 1991 tool guarantees equivalence of the original and the restructured program, but is inefficient.
- Opdyke’s 1992 code refactoring tool is efficient but does not guarantee equivalence.
- MPPR will guarantee equivalence and be efficient enough for programmers to use on a regular basis.
public boolean test(Cmd c, int x, int y) {
    ...; return true; }

Either
- Makes all global replacements necessary to apply the selected command and maintain equivalence
- Or returns “Cannot restructure” error message
MPPR must ensure that the underlying the diverse program representations – source code, abstract syntax tree, program dependence graph, etc. – are kept consistent with each other.

As more restructurings are added and new representations are needed, the underlying abstractions must isolate such changes.

The implementation of these abstractions must be efficient enough to ensure that the end-user is satisfied with the tool’s performance.
Griswold’s 1991 design used layering and achieved the first two desiderata – but inefficiently due largely to passing computations through the layers.

Layering separated the definition of the abstractions from the maintenance of their consistency.

Layering supported creation of subsets and supersets to ease anticipated changes.

Program representations (e.g., the AST and the PDG) encapsulated in modules maintain their independence from one another – a mediator-style pattern separated the management of the relationships among these representations.
Five basic layers (shown horizontally by dotted lines)

Three basic modules (show as columns)

Boxes are submodules; dotted boxes are submodules re-exported from a lower layer

Arrows are calls or event announcements
Representation layer: example

- The AST and the PDG in the representation layers name common variables and operators.
- This relationship is maintained as part of the AST-PDG mapping; for example, $x + 2 \times y$.
Core algorithm: example var-to-expr

```
procedure var-to-expr([v := e])
  for uᵢ ∈ get-uses([v := e]) do
    ast-subset!(ast-copy(e), uᵢ)
    ast-remove!([v := e])
end
```

- This is supplemented by a check to ensure that the restructuring will maintain meaning
  - Scope must be checked
  - No side-effects
  - ...

- These operations all occur in the global transformation and restructuring layers
Entire design of basic MPPR completed
  - Implementation completion expected May 15, 2010 (see schedule on website)

Design of extended architecture, with direct update of PDGs, for efficiency, is underway
  - Backup in case performance of basic design is not fast enough