How to depend on a war-packaged project

The main pattern for having a project depend on another is as follows:

```
define 'problematic' do
  define 'foo' do
    package :jar
  end

  define 'bar' do
    compile.with project('foo')
  end
end
```

The above works perfectly, but it would fail to compile if project 'foo' was packaged as war instead of jar. You might solve the problem with

```
define 'problematic' do
  define 'foo' do
    package :war
  end

  define 'bar' do
    compile.with project('foo').compile.target
  end
end
```

This solves the problem in the compilation task, but fails in the eclipse task, because the generated .classpath would contain an incorrect entry. (This is a bug in the Eclipse task as of 1.4.6). To solve this problem you might do

```
define 'problematic' do
  define 'foo' do
    package :war
  end

  define 'bar' do
    compile.with project('foo'), project('foo').compile.target
    eclipse.exclude_libs += [project('foo').compile.target]
  end
end
```

You interpret this like this: in the compile.with, the first element "project('foo')" is used by the eclipse task to set up the project dependency correctly. The second element is used by the compile task to make foo's classes available. The eclipse.exclude_libs options fixes the bug in the Eclipse task by removing the defective .classpath entry.

If you also want to depend on foo's test classes you might write

```
define 'problematic' do
  define 'foo' do
    package :war
  end

  define 'bar' do
    compile.with project('foo'), project('foo').compile.target, project('foo').test.compile.target
    eclipse.exclude_libs += [project('foo').compile.target, project('foo').test.compile.target]
  end
end
```

that can be made clearer with

```
define 'problematic' do
  define 'foo' do
    package :war
  end

  define 'bar' do
    foo_src = project('foo').compile.target
    foo_test = project('foo').test.compile.target
    compile.with project('foo'), foo_src, foo_test
    eclipse.exclude_libs += [foo_src, foo_test]
  end
end
```