Dependency Architecture

Jim Fawcett

CSE681 – Software Modeling & Analysis

Fall 2002

Use Cases

- Dependency analysis generates information for:
  - Building test plans:
    - Don’t test a module until all the modules on which it depends have been tested.
  - Software maintenance:
    - What modules depend on the module we plan to change? We need to test them after the change to see if they have been adversely affected.
  - Documentation:
    - Documenting dependency information is an integral part of the design exposition.

Scope of Analysis

- This architecture is concerned with ***dependencies*** between a program's ***modules***.
  - A module is a relatively small partition of a program's source code into a cohesive part.
- A typical module should consist of about 400 source lines of code (SLOC).
  - Obviously some will be smaller, some larger, but this is a good target size
- Typical project sizes are:
  - Modest size research project – 10,000 sloc
    $\Rightarrow$ 25 modules
  - Modest size commercial product – 600 kslocs
    $\Rightarrow$ 1,500 modules

Conclusions from Use Case Analysis

- Even for relatively modest sized research projects, there is too much information to do an adequate analysis by hand.
  - We need automated tools.
  - The tools need to show dependencies in both ways, e.g.:
    - What files does this file depend on?
    - What files depend on this file?
  - The tools need to disclose dependencies between all files in the project.

Critical Issues

- Scanning for Dependencies in C# modules
- Data structure used to hold dependencies
- Displaying large amounts of information to user
- False dependencies due to unneeded includes in C++ modules
- Dependence on System Libraries

Dependency Scanning

- Will naïve scanning work for 1500 files?
  - If opening and scanning a single file takes 25 msec, then:
    - Finding dependencies for 1 file takes:
      0.025 X1500 / 60 = 0.625 minutes
    - Finding dependencies for all files takes:
      0.625 X 1500 / 60 = ***15.6 hours***!
- So let's scan each file once and store all its identifiers in hash table in RAM.
  - If that takes 30 msec per file:
    - Then making hash tables for all files takes:
      0.03 X 1500 / 60 = 0.75 minutes
    - If hash table lookup takes 10 μsec per file then finding dependencies between all files takes:
      0.00001 X 1500 X 1500 / 60 + 0.75 = ***1.125 minutes***!

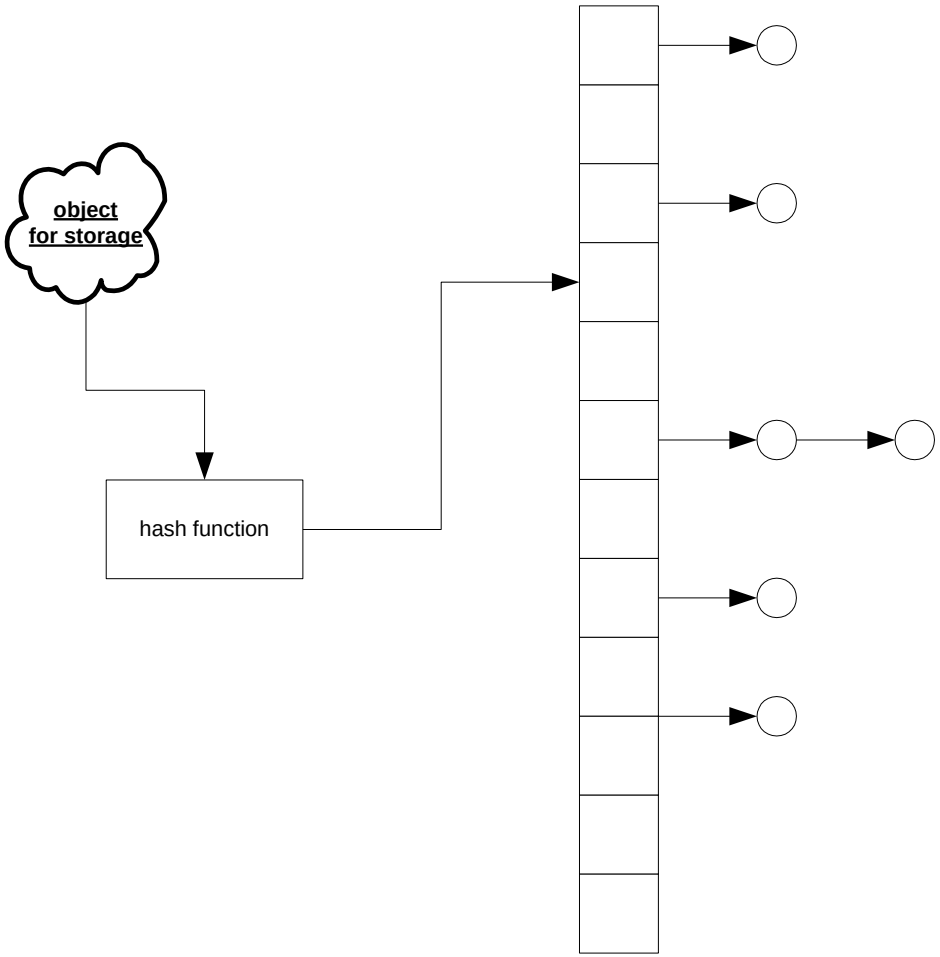
Timing Results Parsing Prototype Source

	Conservative Estimate	Prototype Results
Open file, parse, store in Hashtable – Millisec	25	7
Hashtable Lookup - Microsec	10	0.6

Comparison of Estimated with Measured

- Naïve scanning – scan each file 1500 times:
  - Estimated time to complete scanning of 1500 files:
    15.6 hours
  - Measured time to complete scanning of 1500 files:
    4.4 hours
- Processing each file once and storing in Hashtable, then doing lookups for each file:
  - Estimated time to complete processing:
    1.1 minutes
  - Measured time to complete processing:
    0.2 minutes

Hash Table Layout



Memory to Store Hash Tables

- Assume each file is about 500 lines of source code $\Rightarrow$ about 30 chars X 500 = 15 KB
  - Assume that 1/3 of that is identifiers
    - The rest is comments, whitespace, keywords, and punctuators

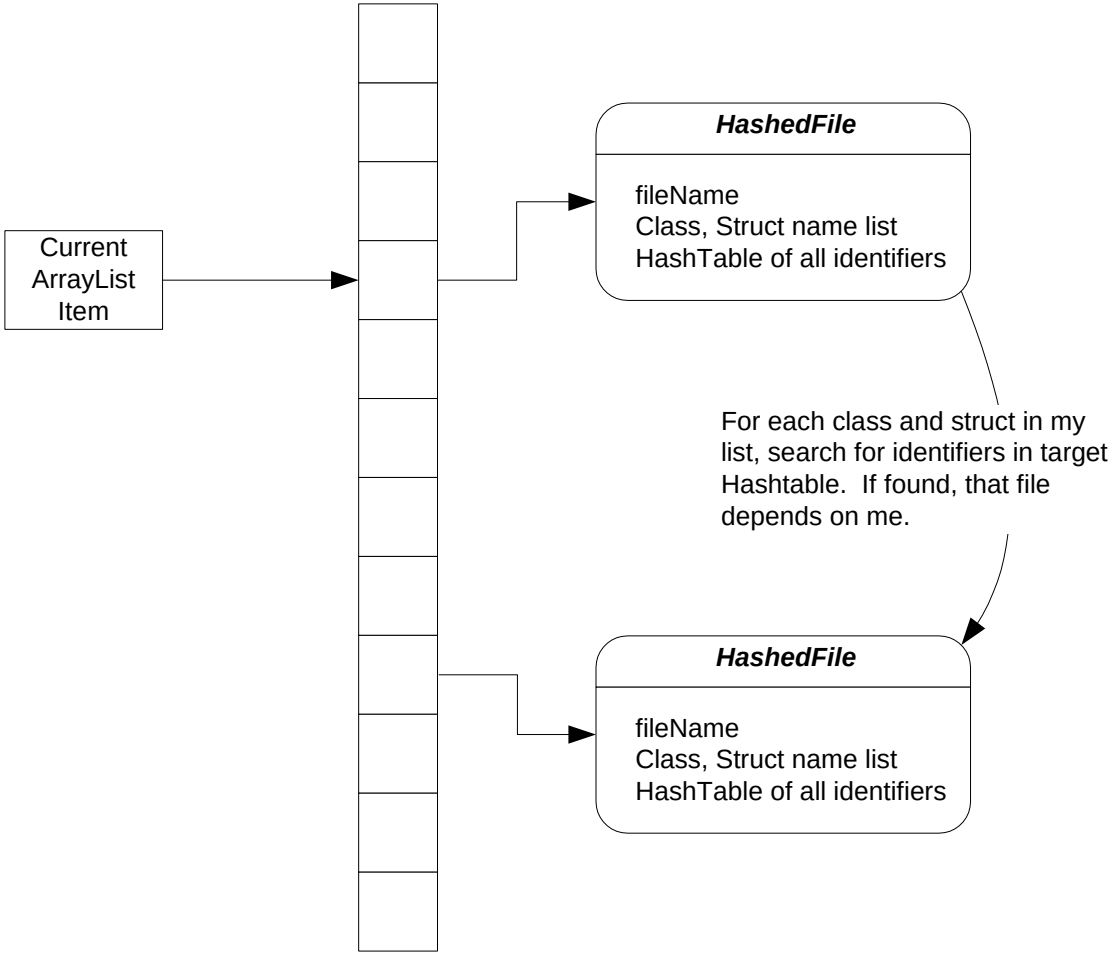
    $\Rightarrow$ 5 KB of indentifier storage
  - Assume HashTable takes 10 KB per file, so the total RAM required for this data is:

    0.01 X 1500 = ***15 MB***.
  - That's large, but acceptable on today's desktop machines.

File Scanning

- For each file in C# file set:
  - For each class and struct identifer in file
    - Look in every other file's HashTable for those identifiers
    - If found, other file depends on current file
  - Record dependency
  - Complexity is $O(n^2)$
- For each file in C++ file set:
  - #include statements completely capture dependency.
  - Record dependency
  - Complexity is O(n)

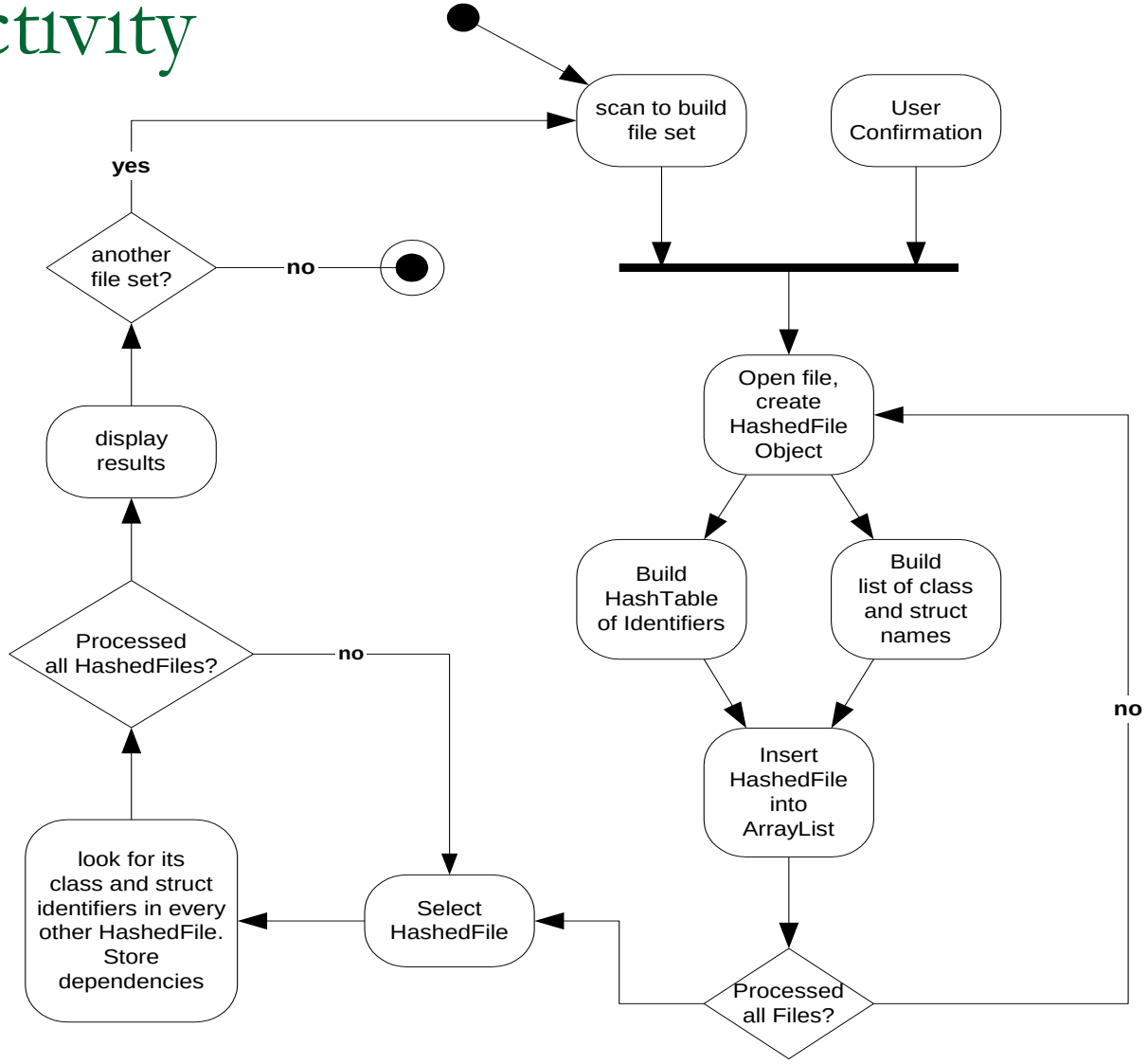
C# Scanning Process



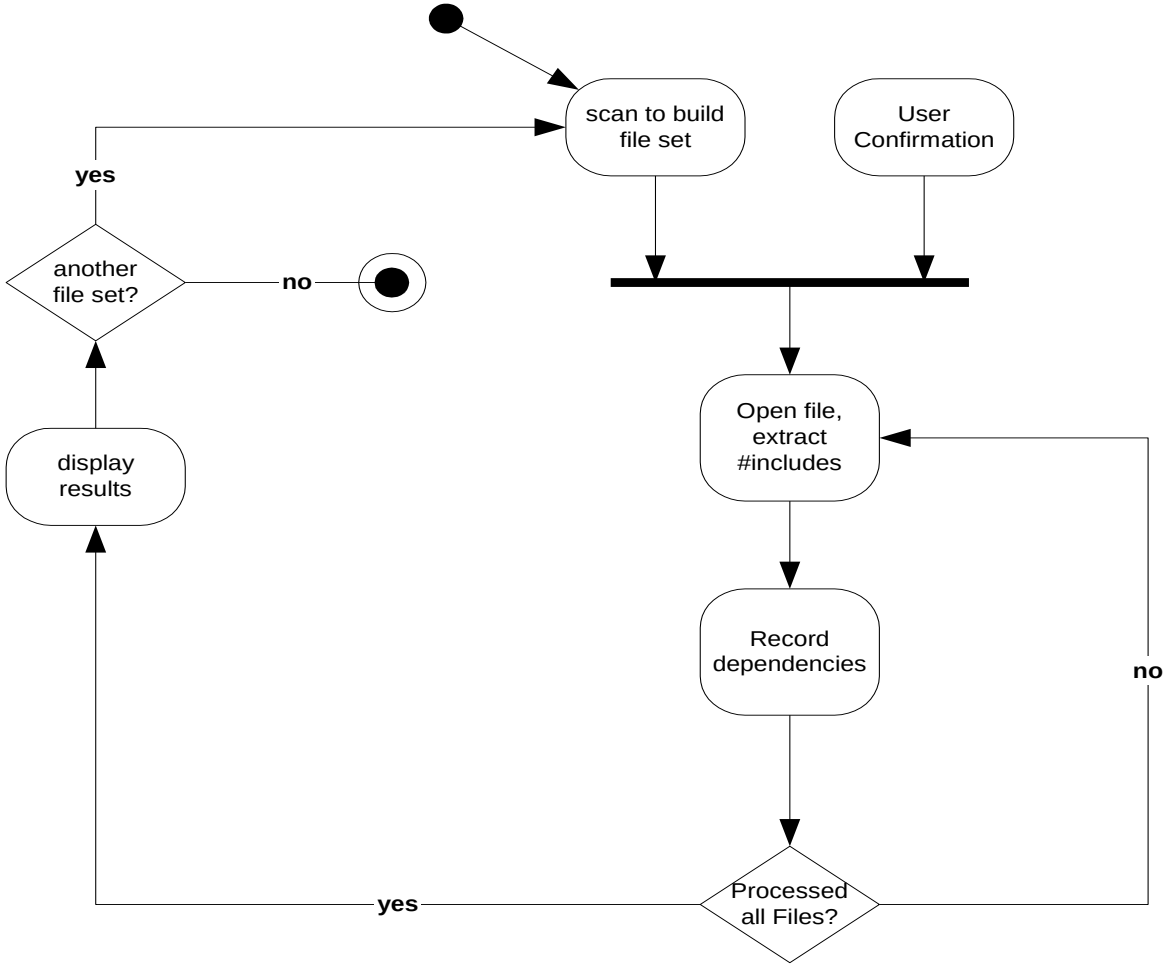
C# Scanning Activities

- Define file set
  - User supplies by browsing, selection, patterns
  - User may wish to scan subdirectory
- Extract token information from each file:
  - Extract tokens from each file and store in HashTable.
  - Save list of Class and Struct identifiers from scan
  - Create HashedFile type with filename, class and struct list, and HashTable as data.
  - Store HashedFiles in ArrayList
- For each HashedFile in list:
  - Walk through ArrayList searching HashTables for the identifiers in class and struct list (note that this is very fast).
  - First time one is found, stop processing file – dependency found.

C# Scan Activity Diagram



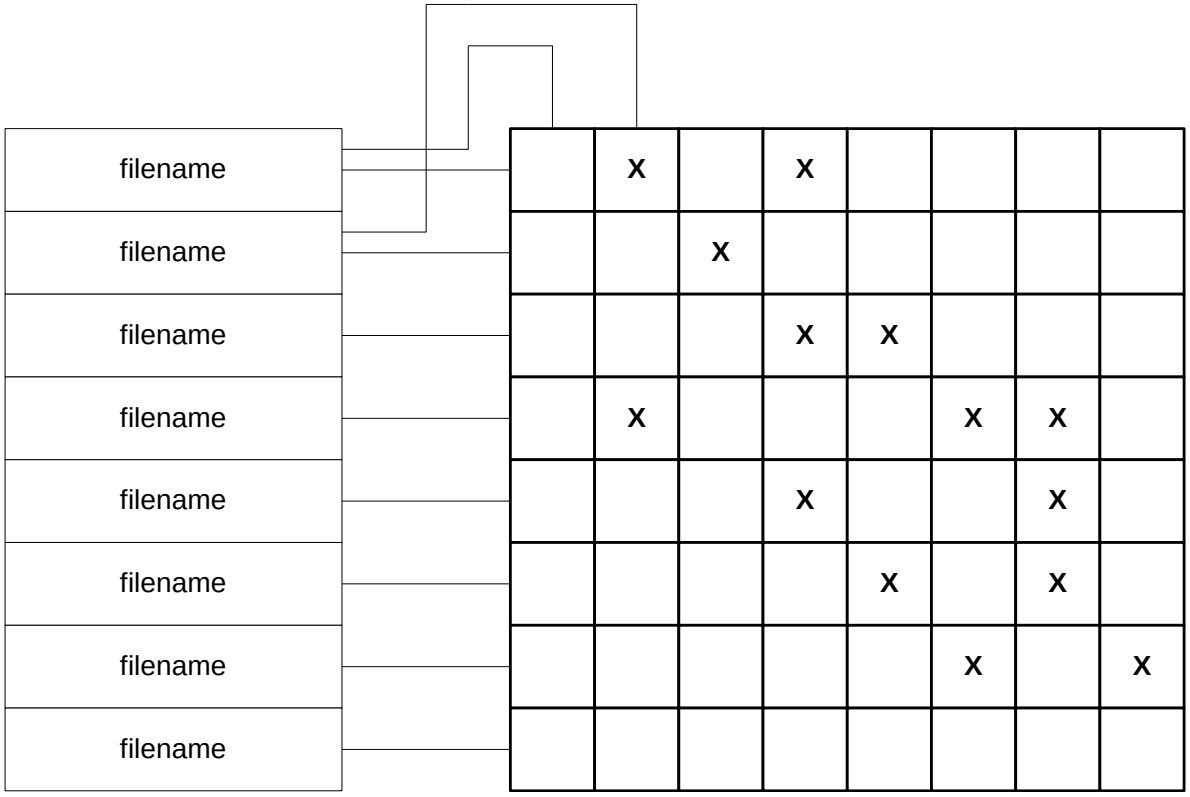
C++ Scan Activity Diagram



Memory to Hold Dependencies

- Naïve storage uses a dense matrix. With 1500 files, that's 2,250,000 elements.
  - Assume each path name is stored only once and we save 75 bytes of path information, so with 1500 files $\Rightarrow$ 112.5 KB
  - Dependency is a boolean and takes 1 byte to store $\Rightarrow$ 2.25 MB.
  - So, the total dependency matrix takes 2.36 MB.
- Therefore, naïve storage is acceptable.

Dependency Matrix



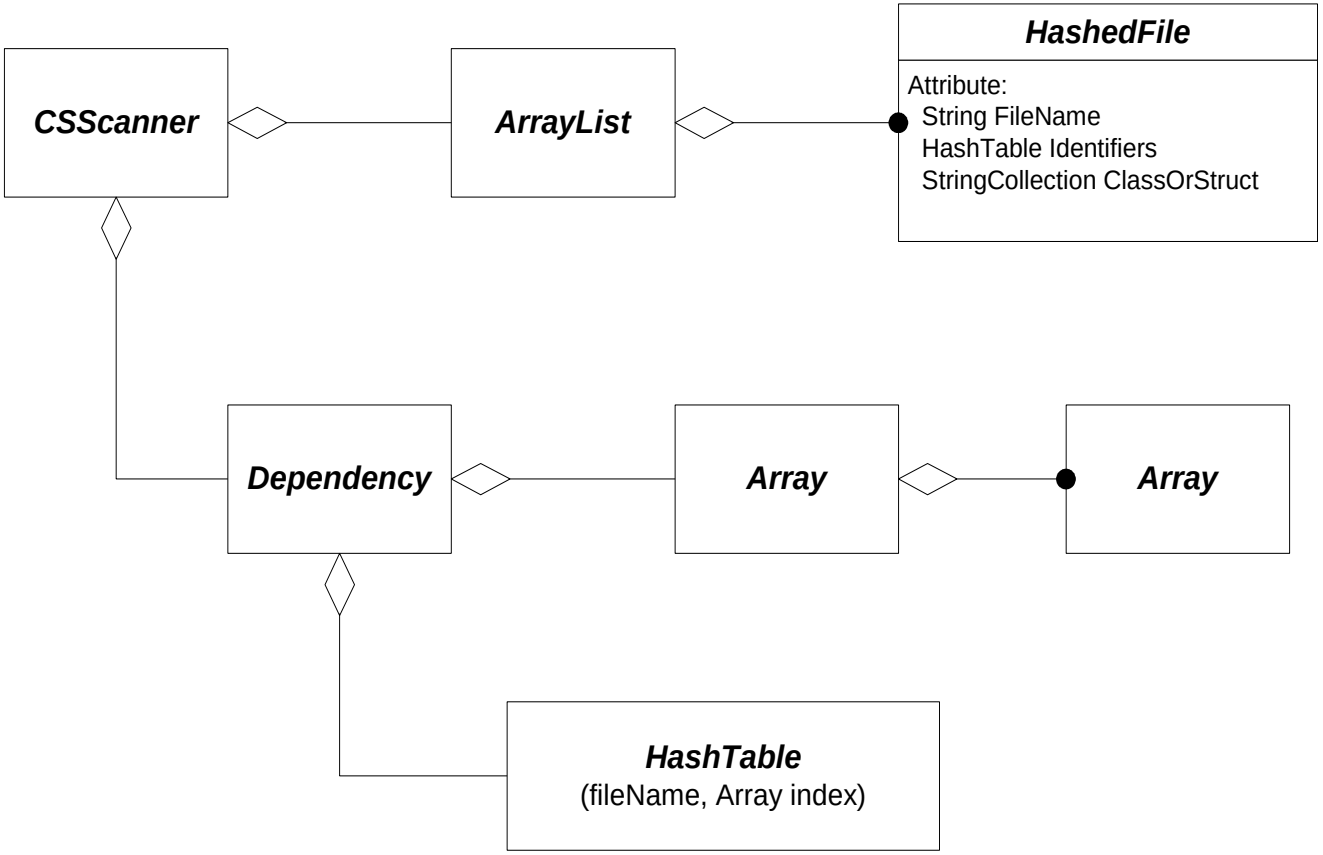
False Dependencies in C++ files

- Need to scan both .h and .cpp files.
- Could programmatically comment out each include – one at a time – and attempt to compile, thus finding the ones actually needed.
  - We would probably do this with a seperate tool.
- We could also just scan, as we do for C#, but that is harder for C++ since we need to check dependencies on global functions and data as well as classes and structs.

Dependence on System Libraries

- Not practical to scan for system dependencies in C#.
  - Can’t find source modules.
  - System dependencies can be found using reflection, but are not particularly useful.
- System dependencies in C++ are easy to find from #include<someSystemHeader>
  - This information is often useful, so why not provide it?

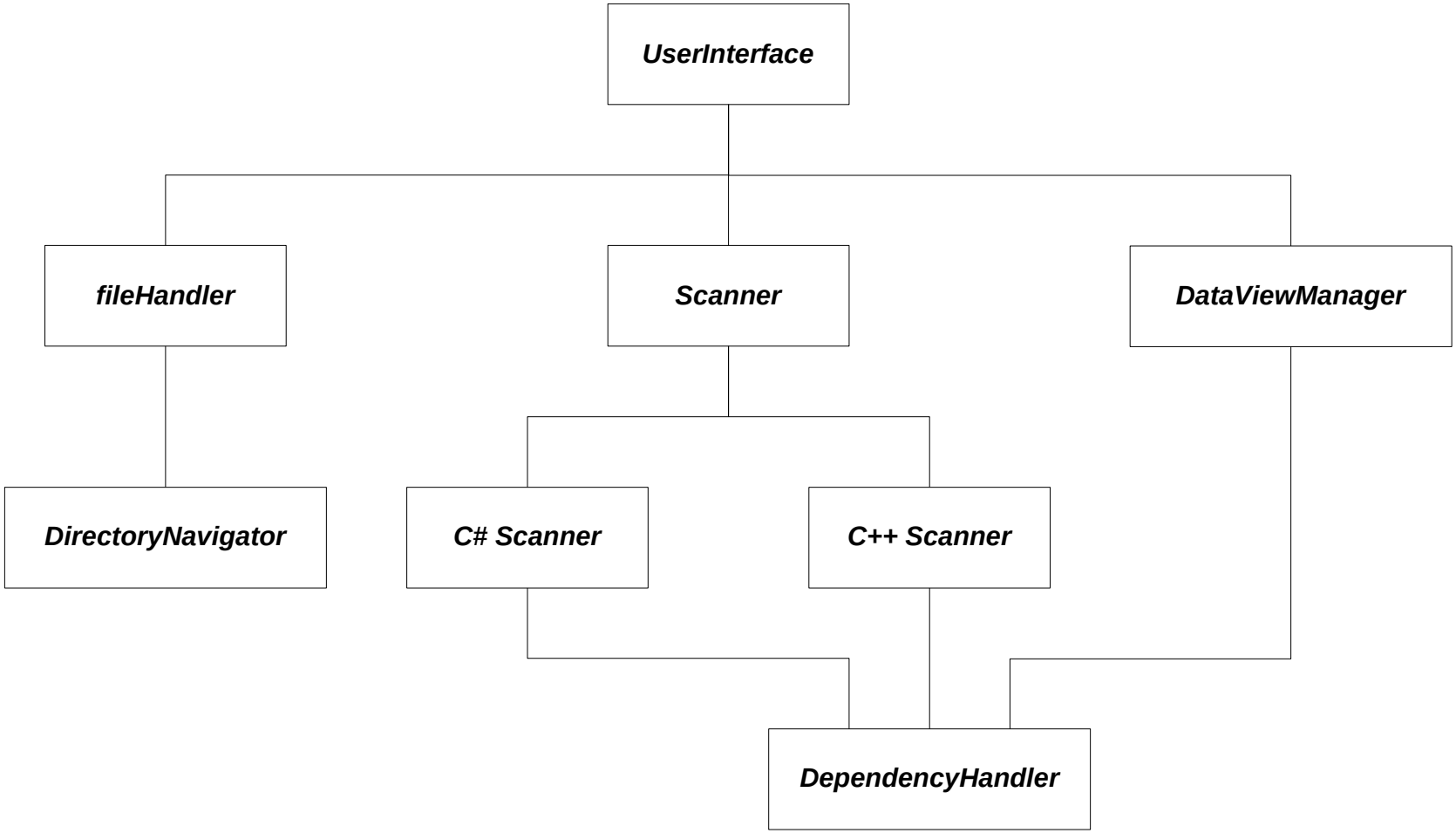
C# Scanner Class Diagram



Displaying Large Sets of Dependencies

- User will probably want to:
  - Enter a name and get list of dependencies.
  - Find all files with no dependencies.
  - Find all files dependent on only the files processed so far this run.
  - Show list of files entered so far and list of files not entered yet.
  - Select subset of files for display.
  - Show a compressed (bitmap?) matrix.
  - Show a scrolling list of files with their dependencies.
  - Show list of names, not matrix row. Matrix row may be far too long to view (e.g., 1500 elements).

Partitions


UserInterface
fileHandler
Scanner
DataViewManager
DirectoryNavigator
C# Scanner
C++ Scanner
DependencyHandler

Summary of Critical Issues

- Scanning for Dependencies in C# modules √
- Data structure used to hold dependencies √
- Displaying large amounts of information ~√
- False C++ dependencies ~√
- Dependence on System Libraries
  - C# X
  - C++ √

Prototype Code

- Scanning – critically important
  - How much time to open file and scan for class, struct identifiers?
  - How much time to build HashTables and HashedFile objects?
  - How much time to evaluate dependencies between two files by HashTable lookup?
- Sizes - important
  - How big is HashedFile object for typical files?
- User Display – could leave to design team with requirement for early evaluation.
  - Mockup display alternatives.