DECOUPLING CHANGE FROM DESIGN

Addressing change by making change within modules easier

Michael VanHilst & David Notkin

University of Washington

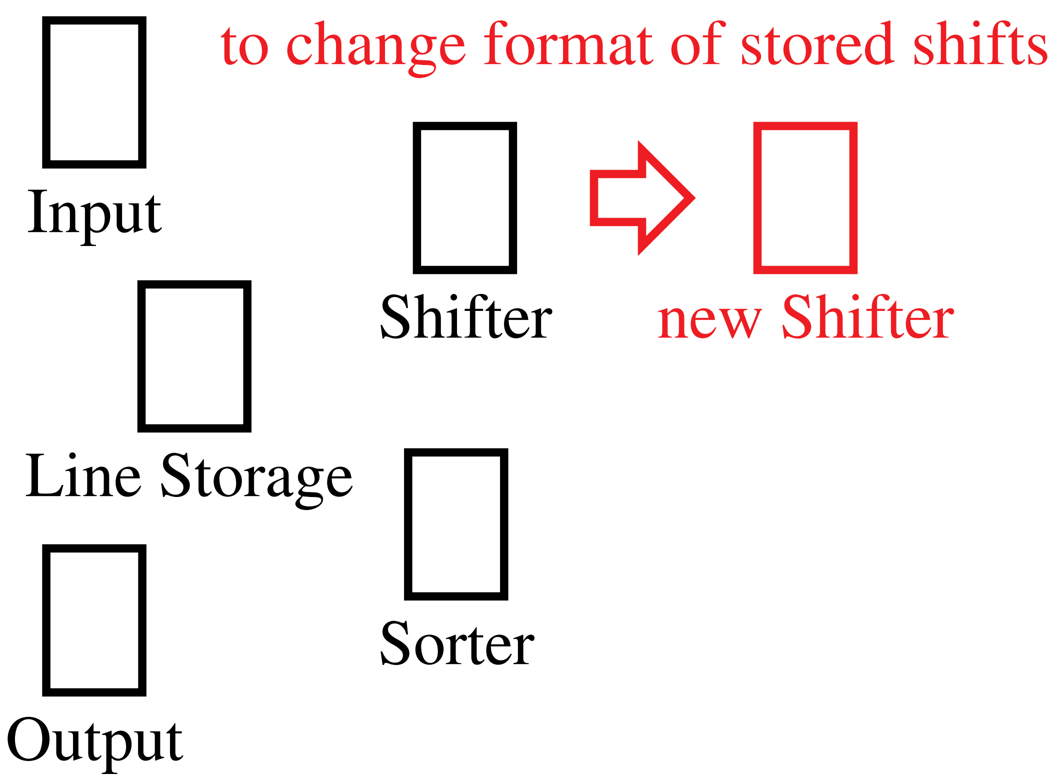
Seattle, Washington, USA

[vanhilst,notkin]@cs.washington.edu

1. **Conventional wisdom**
2. **Alternative approach**
3. **Changes to KWIC**
4. **Broader issues**

MODULARIZING FOR CHANGE

Begin by identifying decisions likely to change. Then design a modularization around those decisions.



Modular design of KWIC

Michael VanHilst and David Notkin
University of Washington

ADDRESSING CHANGE BY BUILDING MODULES AROUND IT...

- **Specifically for anticipated change**

  Forces designer to choose some future changes over others

  It is hard to predict which decisions will change

- **Assumes changing any module is O(1)**

  1000+ line modules are not uncommon

  Within modules multiple decisions are commonly intermingled in the code

INTERMINGLED DECISIONS

```
class ShifterClass {
    LineStoreClass* linestore;
    List<Pair<int>> index;
public:
    void initShifts() {
        index.init();
        for(l=0; l<linestore->numLines(); l++)
            for(w=0; w<linestore->numWords(l); w++)
                index.append(Pair(l, w)); }
    String getLine(int i) {
        Pair p = index.element(i);
        String line = linestore->getLine(p.left);
        return assembleShift(line, p.right); }
    int numLines() { return index.elements(); }
    int numWords() { ... }
};
```

shift algorithm

line source

shift storage

exported interface

MAKING MODULES EASIER TO CHANGE

Decompose modules into submodule components for each decision

Shifter

Within a module submodule components interact through inheritance interfaces

Michael VanHilst and David Notkin
University of Washington

IMPLEMENTING SUBMODULE COMPONENTS

Implement components as *extension layers* using parameterized inheritance

```
template <class SuperType>
class IndexShifter : public SuperType {
public:
    void initShifts() {
        resetShift();
        for(l=0; l<numLines(); l++)
            for(w=0; w<numWords(l); w++)
                addShift(l,w); }
    String assembleShift(int l, int w) { ... }
};
```

IndexShifter component as C++ template

Michael VanHilst and David Notkin
University of Washington

MODULE COMPOSITION

Compose modules by binding inheritance parameters in separate declarations

```
class shifter1 : public GetLineImport<LineStore, obj> {};
class shifter2 : public ShiftIndexStore<shifter1> {};
class shifter3 : public IndexShifter<shifter2> {};
class shifter4 : public GetShiftedLine<shifter3> {};
typedef shifter4 Shifter;
```

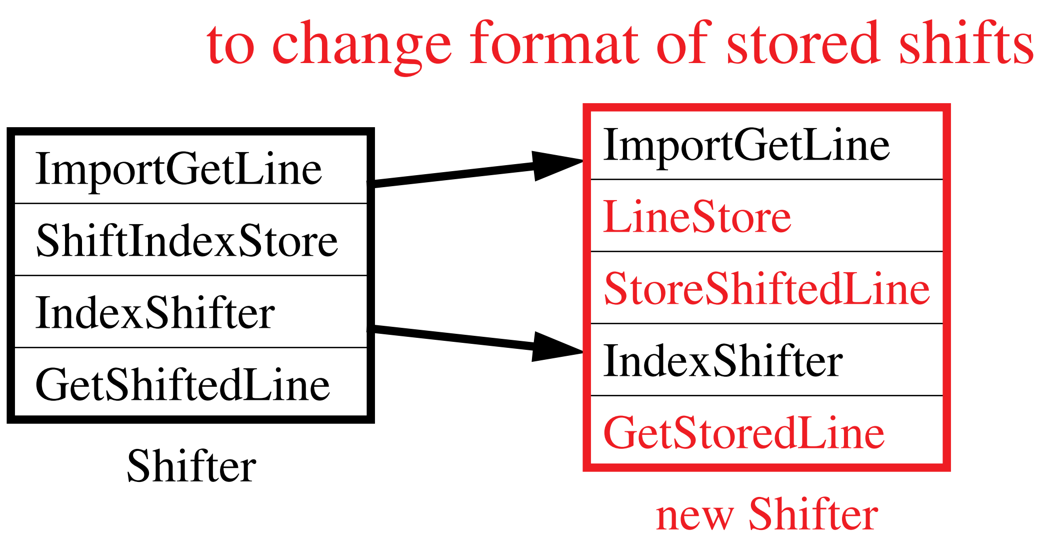
Declarations to compose Shifter module

Composition can be statically type checked and statically optimized

Michael VanHilst and David Notkin
University of Washington

EVOLUTION USING SUBMODULE COMPONENTS

Submodule components narrow the focus of change

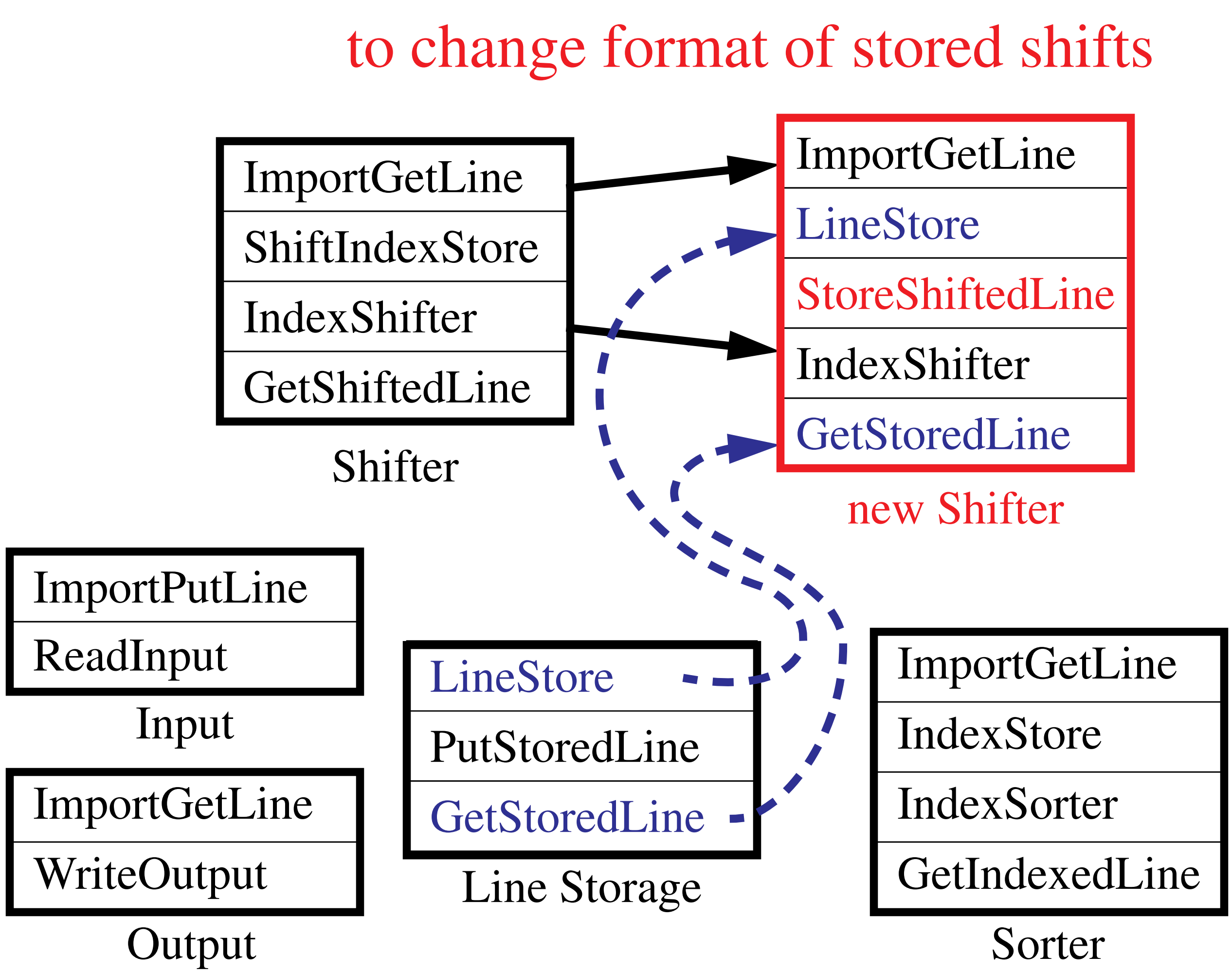


Decomposed KWIC Shifter module and changes to store explicit lines

Michael VanHilst and David Notkin
University of Washington

COMPONENT REUSE

Submodule components narrow the focus of change and support reuse



Decomposed KWIC modules

Michael VanHilst and David Notkin
University of Washington

OTHER DESIGNS

pipe and filters KWIC

LineStore
PutStoredLine
ReadInput
GetStoredLine
ShiftIndexStore
IndexShifter
GetShiftedLine
WriteOutput

Shift Filter

LineStore
PutStoredLine
ReadInput
GetStoredLine
IndexStore
IndexSorter
GetIndexedLine
WriteOutput

Sort Filter

client/ server KWIC

PutLineClient
ReadInput

Input Client

LineStore
PutStoredLine
GetStoredLine
ShiftIndexStore
IndexShifter
GetShiftedLine
IndexStore
IndexSorter
GetIndexedLine
PutLineServer
GetLineServer

KWIC Server

GetLineClient
WriteOutput

Output Client

Michael VanHilst and David Notkin
University of Washington

"WORST CASE" CHANGE?

Reused subcomponents manage change

to change from uncompressed to compressed line storage

ImportLineStore
PutStoredLine
ReadInput

Input

ImportLineStore
GetStoredLine
ImportShiftIndex
IndexShifter

Circular Shift

LineStore

Line Storage

ShiftIndexStore

Shift Index

ShiftIndexStore

Sorted Shift Index

ImportLineStore
GetStoredLine
ImportShiftIndex
ImportShiftIndex
ShiftIndexSorter

Alphabetizing

ImportLineStore
GetStoredLine
ImportShiftIndex
GetShiftedLine
WriteOutput

Output

Functional modularization of KWIC

Michael VanHilst and David Notkin
University of Washington

CHOOSING SUBMODULE COMPONENTS

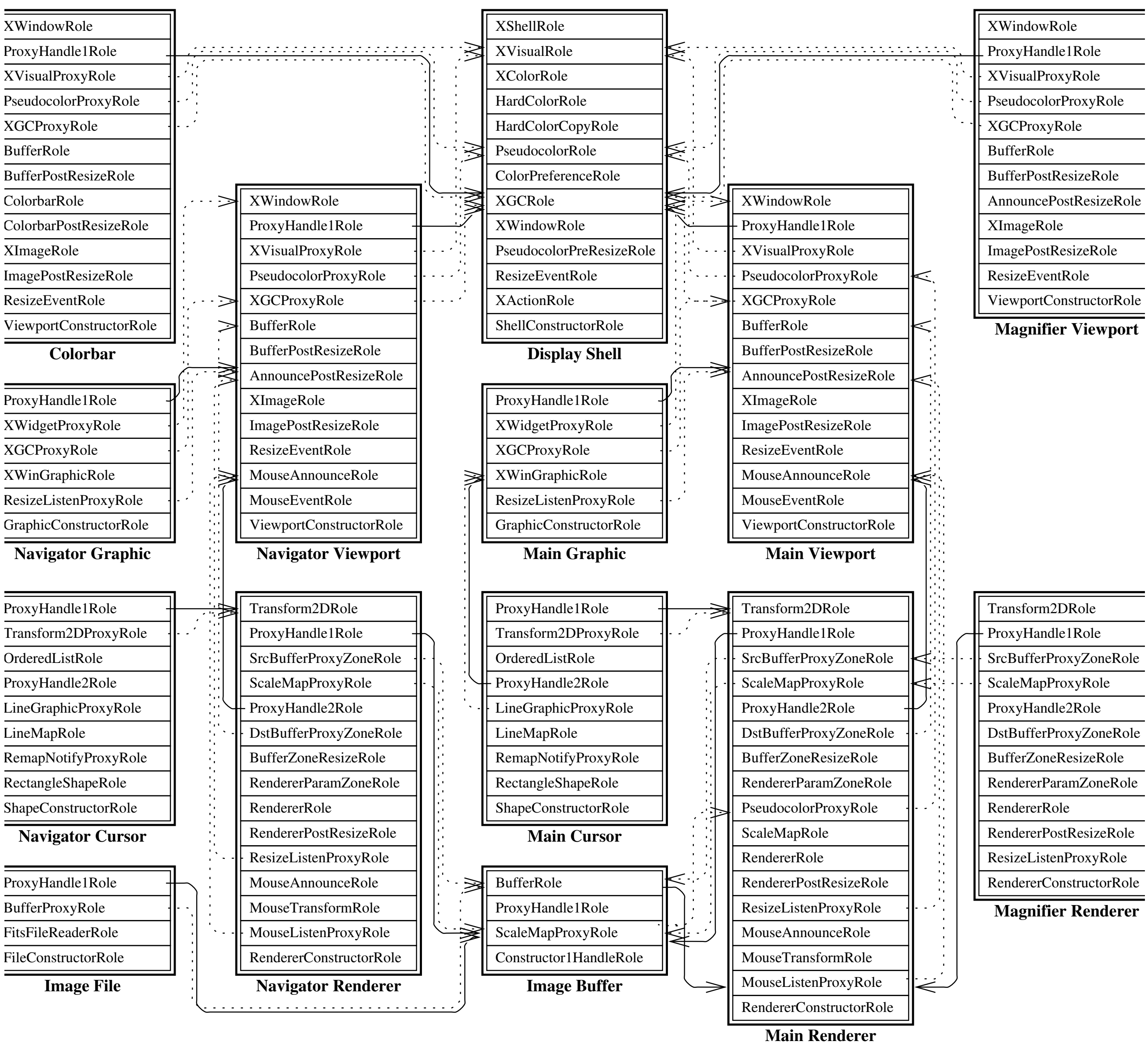
Recursively subdivide components that contain more than one decision

- **order of subdividing is not critical**
- **order of composition is important**
- **guidelines can be used to implement different types of components**

Michael VanHilst and David Notkin
University of Washington

SCALABILITY?

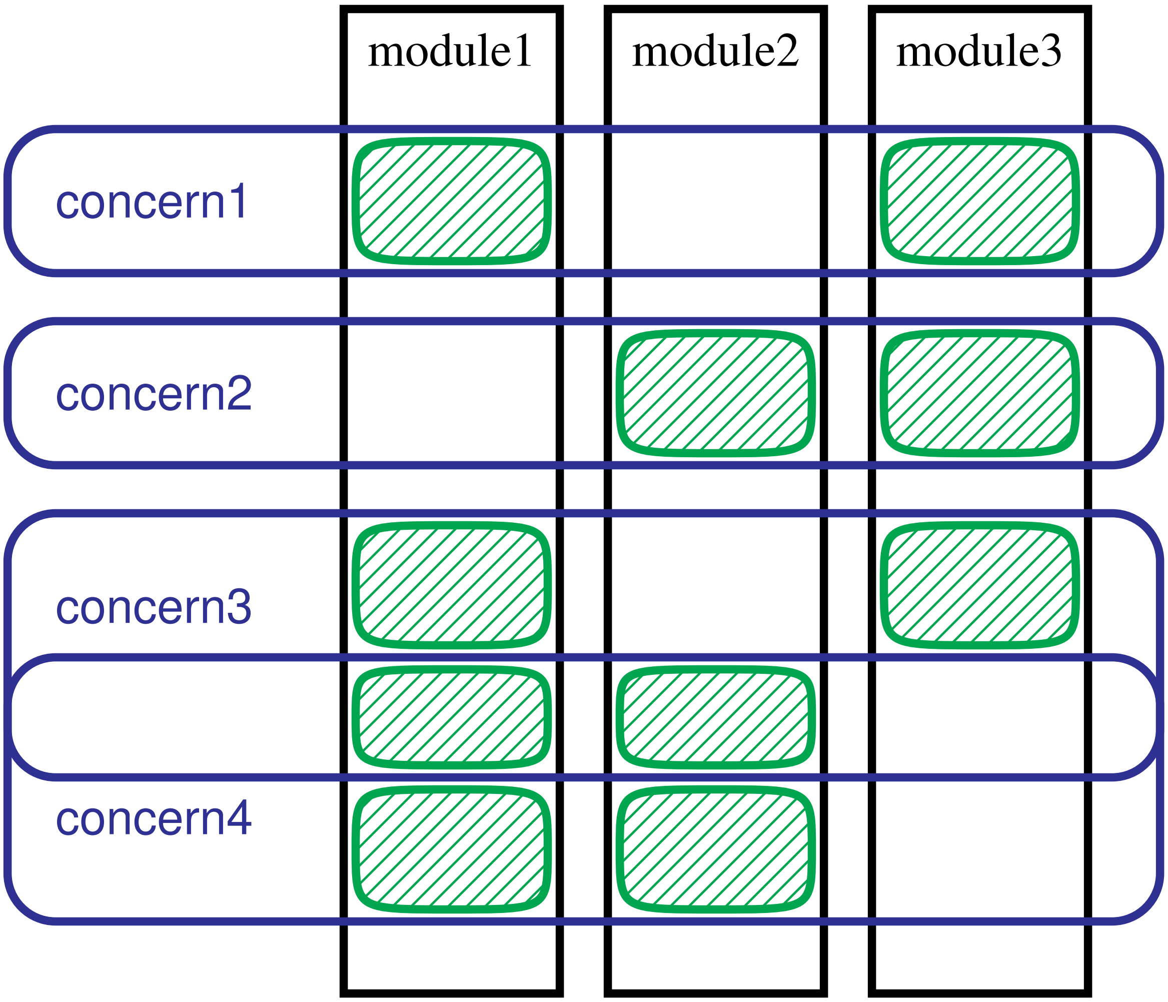
Based on size of module, not application



Michael VanHilst and David Notkin
University of Washington

THE BIG PICTURE

Submodule components capture intersections among modules and concerns



Michael VanHilst and David Notkin
University of Washington

DECOMPOSING MODULES FOR CHANGE

Choose a modularization of the application that is best suited for the problem at hand. Then decompose the modules into submodule components to isolate the decisions that could change.

- **Addresses both anticipated and unanticipated change**

- **Efficient realization using inheritance and parameterization**

Michael VanHilst and David Notkin
University of Washington