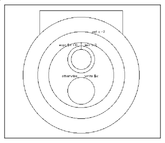
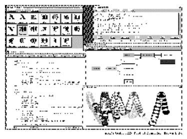
CSE583: Programming Languages

David Notkin
29 February 2000
`notkin@cs.washington.edu`
`http://www.cs.washington.edu/education/courses/583`

Visual programming and program visualization

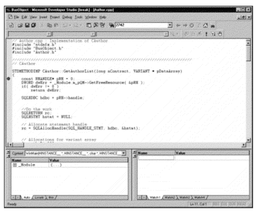




- **In visual programming people use non-textual representations to write programs**
  - **Left: VIPR example while statement [Citrin et al.]**
- **In program visualization people use non-textual representations to understand programs**
  - **Right: Field example screen [Reiss]**

VC++ and similar beasts

- **In general, many of the "visual" programming environments are not supporting visual programming**
- **Rather, they are mostly window-based environments to support textual programming**



BusObject - Microsoft Developer Studio [break] - [Author.cpp]
File Edit View Insert Project Debug Tools Window Help
2742

```
// Author.cpp : Implementation of CAuthor
#include "stdafx.h"
#include "BusObject.h"
#include "Author.h"

/////////////////////////////////////////////////////////////////////////////
// CAuthor

STDMETHODIMP CAuthor::GetAuthorList(long nContract, VARIANT * pDataArray)
{
    const RHANDLE* pRH = 0;
    DWORD dwErr = _Module.m_pQH->GetFreeResource( &pRH );
    if( dwErr != 0 )
        return dwErr;

    SQLHDBC hdbc = pRH->handle;


    //Do the work
    SQLRETURN rc;
    SQLHSTMT hstat = NULL;

    // Allocate statement handle
    rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstat);


    // Allocations for variant array
```

Visual programming

- **People write programs almost solely using text**
- **Rich I/O devices were rare**
- **Technologies (e.g., parsing) were developed to support textual input**
- **"A picture is worth 1000 words"**
- **"Would we not be more productive and would the power of modern computers not be accessible to a wider range of people if we were able to instruct a computer by simply drawing for it the images we see in our mind's eye when we consider the solutions to particular problems?"**
  —M. Boshernitsan

To VP or not to VP?

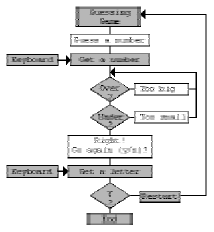
- **In small groups, take about 5-10 minutes to list the top three reasons that**
  - **visual programming *should* (in the long term) dominate textual programming**
  - **textual programming *should* (in the long term) dominate visual programming**
  - **textual programming *does* (now) dominate visual programming**

Does anybody in 584 use VP?

- **If so, how?**

Flowcharts

- A very early visual notation for program
  - Goldstein and von Neumann [1947]
- Well-defined icons
- Supported in
  - physical templates
  - general drawing tools
  - specific flowchart tools (including layout and "generate from code")

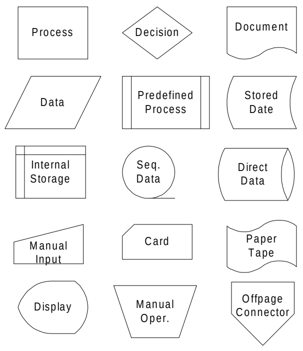


Ainsworth & Partners, Inc.

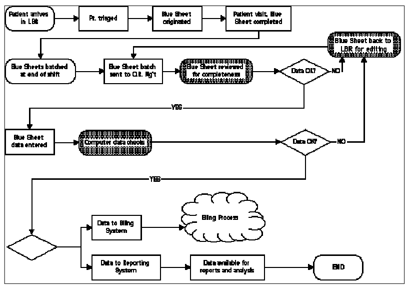
A little more history

- Haibt developed a system that could take Fortran or assembly language programs and generate [1959]
- Knuth developed a system integrating documentation with source code, also automatically generating flowcharts [1963]

Example icons



Robert Luttman & Associates


Patient arrives in LBR
Pt. triaged
Blue Sheet originated
Patient visit, Blue Sheet completed
Blue Sheet back to LBR for editing
Blue Sheets batched at end of shift
Blue Sheet batch sent to Q.R. Mg't
Blue Sheet reviewed for completeness
Data OK?
NO
YES
Blue Sheet data entered
Computer data checks
Data OK?
NO
YES
Data to Billing System
Billing Process
Data to Reporting System
Data available for reports and analysis
END

RFG Quality Consultants

- In part an ISO 9000 consultancy
- "You can use flowcharts to make your quality system more user-friendly: they say a picture is worth a thousand words! A flowchart has a major advantage over written procedures, because it is gives an immediate overview of the method required to the person reading it. It is also usually better to look at and often takes up less pages than its written equivalent."

FlowLynx, Inc. [1998-99]

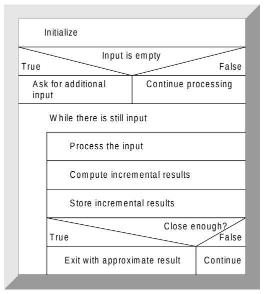
- "Visual FlowCoder (VFC) provides a high performance flowchart browser and editor that frees you from the drudgery of working directly within a text code editor. Flowchart enhanced source code makes any code (yours or someone else's) significantly faster to understand, navigate, learn, reuse, re-engineer and edit. You'll find that Visual FlowCoder delivers the most intense visual programming experience that you've ever seen!"

"Key benefits" include

- Visually documents code so anyone can understand it
- Allows concurrent work on the flow and the code ...
- Makes it easy to understand and optimize machine generated code
- Flowchart any code using only eleven symbols - short learning curve
- Flowcode thousands of lines of code in seconds
- Designed for optimal speed when working with thousand object flowcharts
- Helps the entire programming team understand and share code
- Places all programming languages on an equal visual footing making it just as easy to learn many languages
- Helps engineers optimize their code by highlighting iteration and logical processing
- Visually enhances mining and retrieval of information from legacy code

Nassi-Shneiderman diagrams (1973)

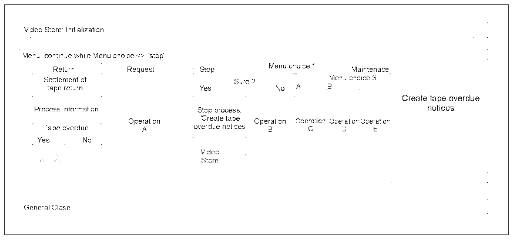
- Intended to assist in defining procedures that capture algorithms
- A graphical representation of a structured flowchart
  - Essentially, flowcharts meet structured programming
- Also called program structure diagrams



Potential benefits

- Debugging, self-documentation and maintenance
- The scope of iteration and of conditionals is well-defined and visible
- The conditions embedded within compound conditions can be seen easily
- The scope of local and global variables is obvious
- Arbitrary transfers of control are impossible
  - Based on Böhm & Jacopini
- Complete structures should fit on one page (with no off-page connectors)
- Recursion has a trivial representation

A larger example



http://wwwis.cs.utwente.nl:8080/dmrg/MEE/misop013/index.html

Flowcharts redux

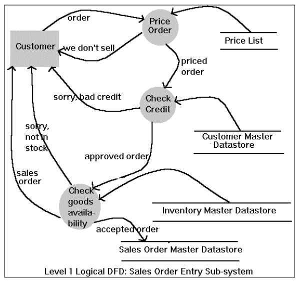
- Flowcharts surely do not satisfy most of the claims in the previous slides
- At the same time, they are surely useful in some situations
  - We do find them scrawled on whiteboards now and then
- But it's not clear at all that they can be used in any direct way to actually effectively write software

Related diagrams

Other diagrams?

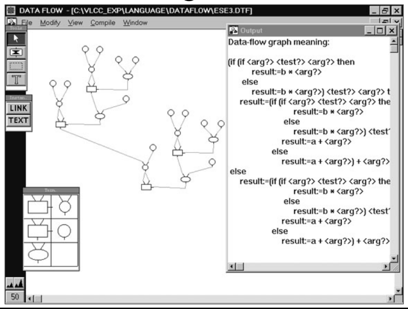
- Dataflow diagrams
- UML diagrams
  - General purpose concepts/use-case diagram
  - Class diagram- types of classes, relationships, visibility & properties
  - State-transition diagram- states & transitions, nested states
  - Sequence diagram
  - Collaboration diagram
  - Component/deployment diagrams

Dataflow diagrams


order
Price Order
Price List
Customer
we don't sell
priced order
sorry, bad credit
Check Credit
sorry, not in stock
Customer Master Datastore
approved order
sales order
Check goods availa-bility
Inventory Master Datastore
accepted order
Sales Order Master Datastore
Level 1 Logical DFD: Sales Order Entry Sub-system

Gangolly,
U. Albany

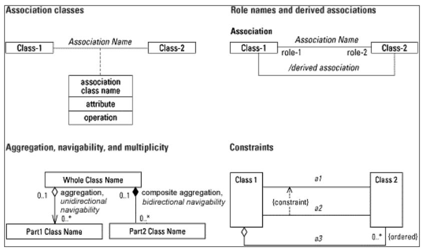
Dataflow diagram



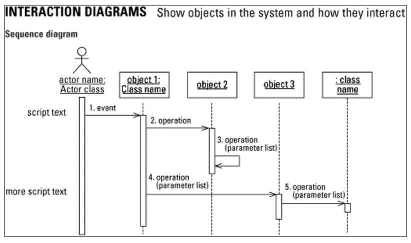
```
Data-flow graph meaning:

(if (if <arg?> <test?> <arg?> then
      result:=b * <arg?>
   else
      result:=b * <arg?>) <test?> <arg?> t
   result:=(if (if <arg?> <test?> <arg?> the
              result:=b * <arg?>
           else
              result:=b * <arg?>) <test'
           result:=a + <arg?>
        else
           result:=a + <arg?>) + <arg?>
else
   result:=(if (if <arg?> <test?> <arg?> the
              result:=b * <arg?>
           else
              result:=b * <arg?>) <test'
           result:=a + <arg?>
        else
           result:=a + <arg?>) + <arg?>
```

University
of Pisa

UML class diagram example
(rational.com)



UML sequence diagram


INTERACTION DIAGRAMS Show objects in the system and how they interact
Sequence diagram
actor name: Actor class
object 1: Class name
object 2
object 3
: class name
script text
1. event
2. operation
3. operation (parameter list)
more script text
4. operation (parameter list)
5. operation (parameter list)

A little VP history

- Sketchpad [Sutherland 63]
  - A simple constraint-based graphics system
  - Followup work by W. Sutherland
    - Visual creation, debugging, execution of dataflow diagrams
- Pygmalion [D.Smith 75]
  - Attempt to allow programming that corresponded to the creative thought processes
  - Icon-based programming paradigm
  - Essentially “programming-by-example” to generate text programs

vs. conventional languages

- In conventional languages, tokens are concatenated to form a program
- In VPLs, icons correspond to tokens
  - But construction rules are explicit
    - horizontal concatenation
    - vertical concatenation
    - spatial overlay
- Analyze these programs
  - Using picture grammars, graph grammars, …
  - The result is parse/abstract syntax trees
  - The compiler works symbolically, not with icons

Taxonomy [Chang, Shu, Burnett]

- Pure visual language systems
  - Graphical representations only for creation, manipulation, execution, debugging
  - VIPR, Prograph, PICT/D, Cube, …
- Hybrid languages
  - Create programs visually and then manipulate textually
  - Or add graphical elements to a textual language
  - Rehearsal World, C2, work by Erwig, …

More taxonomy

- Programming by example systems
  - Rehearsal World, Pygmalion, …
- Constraint-oriented and physical simulation systems
  - ThingLab, ARK, …
- Form-based languages
  - Use a spreadsheet metaphor
  - Forms/3, …

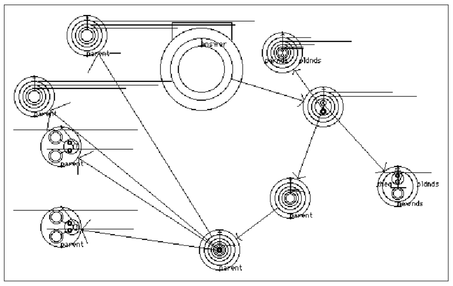
VIPR

- Visual Imperative Programming
  - Citrin et. al at the U. Colorado
- Intended for completely visual general purpose programming
- Uses nested series of concentric rings to visualize programs
  - Instead of icons, forms or other traditional graphical representations

Network of pipes

- Each step in a computation merges two rings in the presence of a state object that is connected to the outermost ring
- Walk down a network of pipes that branches off in different directions, changing the state based on actions written on the inside of the pipes

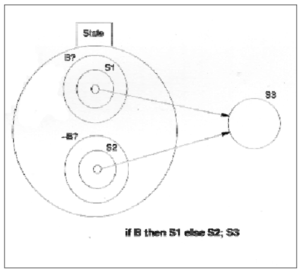
VIPR program



Motivation and semantics

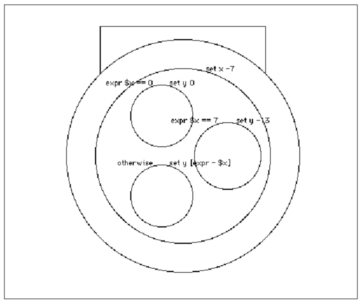
- Create an OO language that is relatively easy to learn and use
- VIPR includes most of OO constructs, including inheritance, polymorphism and dynamic dispatch
- Has relationship both to C++ semantics and also to simple (λ-calculus based rewriting rules)

VIPR if-then-else



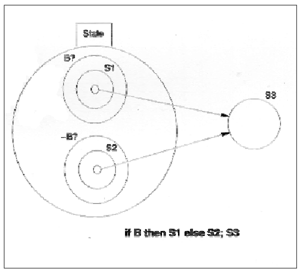
University of Washington • CSE583 • D. Notkin © 2000 31

VIPR case statement



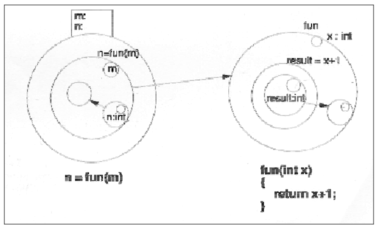
University of Washington • CSE583 • D. Notkin © 2000 32

VIPR while



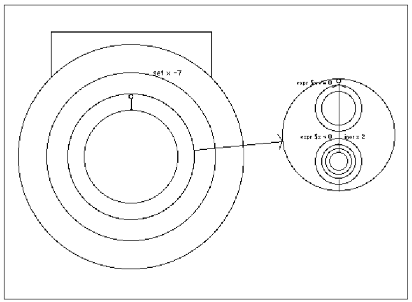
University of Washington • CSE583 • D. Notkin © 2000 33

VIPR function call and return



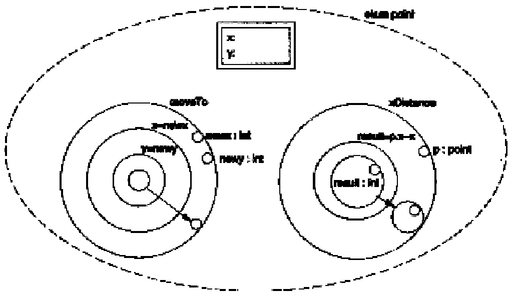
University of Washington • CSE583 • D. Notkin © 2000 34

VIPR recursive call



University of Washington • CSE583 • D. Notkin © 2000 35

VIPR class definition



University of Washington • CSE583 • D. Notkin © 2000 36

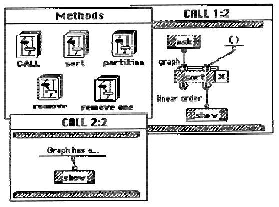
ARK (Alternate Reality Kit) [R. Smith 86-92]

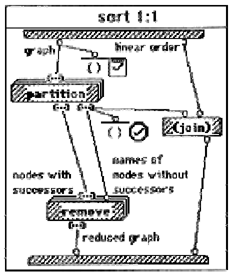
- A 2D animated environment for creating interactive simulations
- The goals were
  - to teach users about fundamental laws of physics
  - to allow non-expert programmers to develop interactive simulations
- Objects have visual representation, mass and velocity
- Laws of nature are objects that can be manipulated and changed
  - Very much like the meta-object protocol

Prograph [Pietryzkowski & Cox/ Pictorius]

- OO pictorial programming environment
- Describe procedures as control-flow diagrams and method invocation as pattern-matching
  - low-level programming using method definitions
  - high-level programming by combining methods into classes and then hierarchies of classes (libraries)
- Each ADT encapsulated in a class
  - objects instantiated from classes

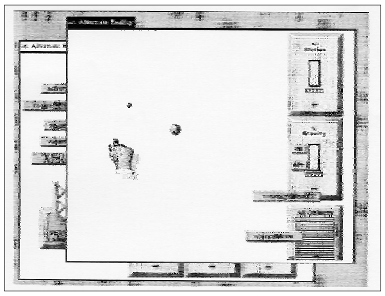
Prograph examples





Topological sort
(method definition on right)

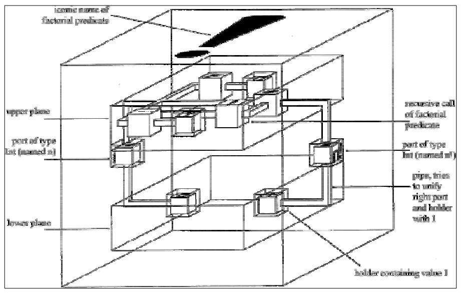
ARK: planetary orbit simulation



Cube [Najork]

- First 3D VPL, using dataflow
- 3D allows more information in an environment to be displayed in given screen size
- Cube programs are composed of holder cubes, predicate cubes, definition cubes, ports, pipes and planes

Cube example


iconic name of factorial predicate
upper plane
port of type list (named n)
lower plane
recursive call of factorial predicate
port of type list (named n')
pipe, tries to unify right port and holder with 1
holder containing value 1

Example explanation (factorial)

- Ports represent input/output
  - Left-hand is “input”; right-hand is “output”
  - Ports are bidirectional for constraint-like computations
- Holder cubes contain data
- Ports connected through pipes to holder cubes
- Each plane is a dataflow diagram
  - The bottom plane represents the recursive base case, with default values for ports and indications of type

More explanation

- If the value at the input port is 0, then the bottom plane is active and the value 1 flows to the output port
- If the input > 0, then 1 is subtracted from the input by the bottom branch of the upper dataflow diagram
- This result is fed to the recursive call to factorial, multiplying the original input by the result
- The product flows to the output port

Program visualization

- Use visualization to understand (as opposed to manipulate) programs
- "The purpose of computing is insight, not numbers" [Hamming 62]

Why use visualization [cs.arizona.edu]

- Physical size needs scaling for comprehension
- Time scale needs changing for comprehension
- Features need to be emphasized or de-emphasized
- Interpretation is needed for comprehension
- Subject hidden from view
- Subject not in visible spectrum
- Subject is not physical in nature
- Subject is imaginary

Program visualization

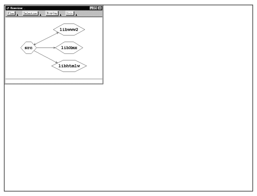
- Scientific and engineering visualizations
- Algorithm animation
  - Algorithms in action
- Software visualization
  - Focus on the structure of software

Algorithm animation

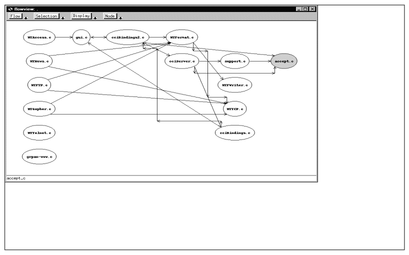
- 1966 Knowlton, Bell Labs, Animation of linked-list language
- 1981 "Sorting our Sorting" Baecker, U. Toronto
- 1984 Balsa, M. Brown, Brown U., full-fledged algorithm animation system
- 1988 UW Illustrating Compiler R. Henry, UW, automatic insertion of animation code during compilation
- 1990 Tango, John Stasko, Brown U., full-fledged algorithm animation system
- 1991 Zeus (successor to Balsa)
- 1993 Polka, J. Stasko, Georgia Tech (extension of Tango to parallel computation)

Visualization

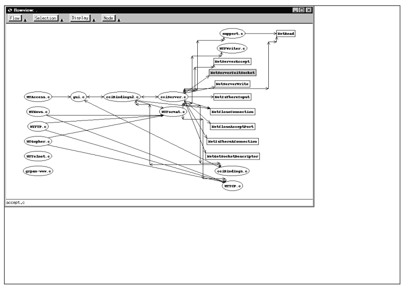
- Pecan, Field, Plum, Imagix 4D, McCabe, etc.
  - Field's flowview is used here and on the next few slides...)
- Provide a graphical "unparsing" of aspects of a software system
- Note: several of these are commercial products



Visualization...



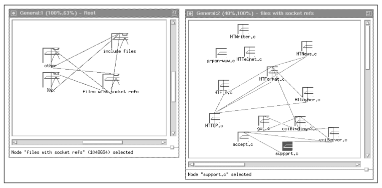
Visualization...



Visualization...

- Provides a "direct" view of the source code
  - Or of an extracted model of the source code
- View often contains too much information
  - So, use elision
  - With elision you usually describe what you are not interested in, as opposed to what you are interested in
    - Some work in fish-eye views helps reduce this problem

Reverse engineering

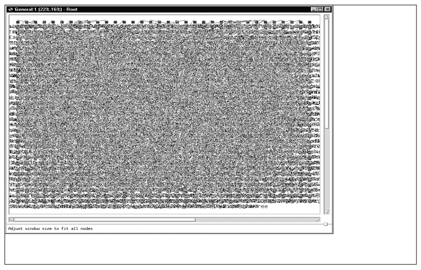


Node "files with socket refs" (1048694) selected

Node "support.c" selected

- Rigi, various clustering algorithms (Rigi is used above)
  - `http://www.rigi.csc.uvic.ca/rigi/rigiframe1.shtml`

Reverse engineering...

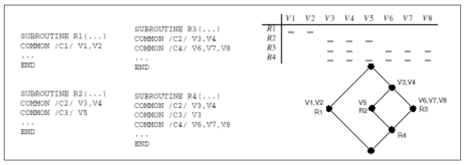


Clustering

- The basic idea is to take one or more models of the code and find appropriate clusters that might indicate "good" modules
  - Coupling and cohesion are at the heart of most clustering approaches
- Many different algorithms

Mathematical concept analysis

- Define relationships between (for instance) functions and global variables [Snelting et al.]
- Compute a concept lattice capturing the structure
  - "Clean" lattices = nice structure
  - "ugly" ones = bad structure

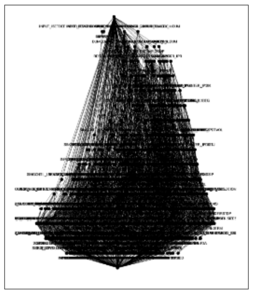


```
SUBROUTINE R1(...)
COMMON /C1/ V1,V2
...
END

SUBROUTINE R2(...)
COMMON /C2/ V3,V4
COMMON /C3/ V5
...
END

SUBROUTINE R3(...)
COMMON /C2/ V3,V4
COMMON /C4/ V6,V7,V8
...
END

SUBROUTINE R4(...)
COMMON /C2/ V3,V4
COMMON /C3/ V3
COMMON /C4/ V6,V7,V8
...
END
```

	V1	V2	V3	V4	V5	V6	V7	V8
R1	–	–						
R2			–	–	–			
R3			–	–		–	–	–
R4			–	–	–	–	–	–

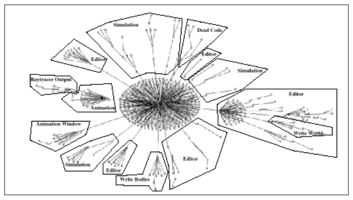
An aerodynamics program

- 106KLOC Fortran
- 20 years old
- 317 subroutines
- 492 global variables
- 46 COMMON blocks



Dominator clustering [Girard & Koschke]

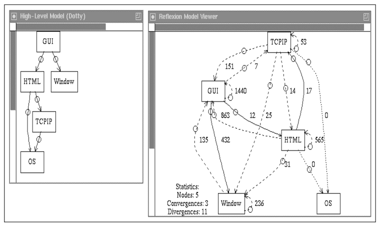
- Rigid body simulation; 31KLOC of C code; 36 files; 57 user-defined types; 480 global variables; 488 user-defined routines



Automatic clustering

- Automatic clustering approaches must try to produce "the" design
  - One design fits all
- User-driven clustering may get a good result
  - May take significant work (which may be unavoidable)
  - Replaying this effort may be hard
- Tunable clustering approaches may be hard to tune
- Unclear how well automatic tuning works

Summarization



Statistics:
Nodes: 5
Convergences: 2
Divergences: 11

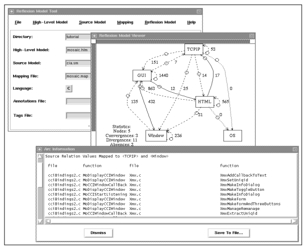
- e.g., software reflexion models

Summarization...

- A map file specifies the correspondence between parts of the source model and parts of the high-level model

```
[ file=HTTCP       mapTo=TCPIP ]
[ file=^SGML       mapTo=HTML ]
[ function=socket  mapTo=TCPIP ]
[ file=accept      mapTo=TCPIP ]
[ file=cci         mapTo=TCPIP ]
[ function=connect mapTo=TCPIP ]
[ file=Xm          mapTo=Window ]
[ file=^HT         mapTo=HTML ]
[ function=.*      mapTo=GUI ]
```

Summarization...



Directory: tutorial
High-Level Model: mosaic.hlm
Source Model: cia.sm
Mapping File: mosaic.map
Language: C
Annotations File:
Tags File:

Statistics:
Nodes: 5
Convergences: 3
Divergences: 11
Absences: 2

```
Source Relation Values Mapped to <TCPIP> and <Window>

 file           function            file                                     function

 cciBindings2.c MoDisplayCCIWindow  Xmx.c                                    XmxAddCallbackToText
 cciBindings2.c MoDisplayCCIWindow  Xmx.c                                    XmxSetUniqid
 cciBindings2.c MoCCIWindowCallBack Xmx.c                                    XmxMakeInfoDialog
 cciBindings2.c MoDisplayCCIWindow  Xmx.c                                    XmxMakeToggleButton
 cciBindings2.c MoCCIStartListening Xmx.c                                    XmxMakeInfoDialog
 cciBindings2.c MoDisplayCCIWindow  Xmx.c                                    XmxMakeForm
 cciBindings2.c MoDisplayCCIWindow  Xmx.c                                    XmxMakeFormAndThreeButtons
 cciBindings2.c MoDisplayCCIWindow  Xmx.c                                    XmxManageRemanage
 cciBindings2.c MoCCIWindowCallBack Xmx.c                                    XmxExtractUniqid
```

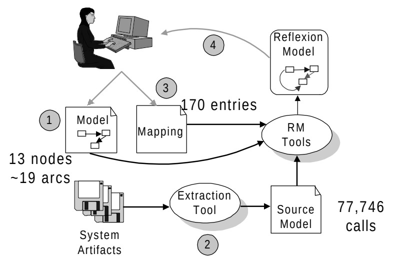
Summarization...

- Condense (some or all) information in terms of a high-level view quickly
- Use a high-level view selected by the programmer
- Some evidence that it scales effectively

Case study: A task on Excel

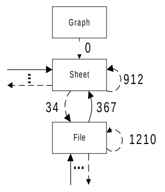
- A series of approximate tools were used by a Microsoft engineer to perform an experimental reengineering task on Excel
- The task involved the identification and extraction of components from Excel
- Excel comprises about 1.2 million lines of C source
  - About 15,000 functions spread over ~400 files

The process used

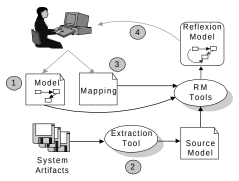


An initial Reflexion Model

- The initial Reflexion Model computed had 15 convergences, 83, divergences, and 4 absences
- It summarized 61% of calls in source model

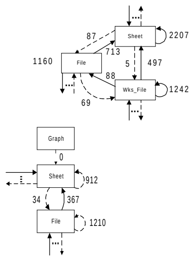


An iterative process



- Over a 4+ week period
- Investigate an arc
- Refine the map
  - Eventually over 1000 entries
- Document exceptions
- Augment the source model
  - Eventually, 119,637 interactions

A refined Reflexion Model



- A later Reflexion Model summarized 99% of 131,042 call and data interactions
- This information was used to reason about, plan and automate portions of the task

Results

- Microsoft engineer judged the use of the Reflexion Model technique successful in helping to understand the system structure and source code
- "Definitely confirmed suspicions about the structure of Excel. Further, it allowed me to pinpoint the deviations. It is very easy to ignore stuff that is not interesting and thereby focus on the part of Excel that I want to know more about." — Microsoft A.B.C. (anonymous by choice) engineer

Learning styles

- An apparent aside

Learning styles [Felder & Solomon]

- Different people learn in different ways
  - At least four identifiable dimensions
  - `http://www2.ncsu.edu/unity/lockers/users/f/felder/public/ILSdir/styles.htm`
- Significant research has been done on these styles
- It isn't that one style is better or worse
  - It may be that technical and engineering fields are *somewhat* self-selective in terms of learning styles

Active vs. reflective learners

- "Active learners tend to retain and understand information best by doing something active with it--discussing or applying it or explaining it to others"
  - [*Electrifying* program representations]
- "Reflective learners prefer to think about it quietly first"

Sensing vs. intuitive learners

- “Sensing learners tend to like learning facts”
- “Intuitive learners often prefer discovering possibilities and relationships”

Sequential vs. global learners

- “Sequential learners tend to gain understanding in linear steps, with each step following logically from the previous one”
- “Global learners tend to learn in large jumps, absorbing material almost randomly without seeing connections, and then suddenly `getting it’”

Visual vs. verbal learners

- “Visual learners remember best what they see--pictures, diagrams, flow charts, time lines, films, and demonstrations”
- “Verbal learners get more out of words--written and spoken explanations”

Results for: David Notkin

```
ACT  X                                                    REF
     11   9   7   5   3   1   1   3   5   7   9   11

SEN                                       X               INT
     11   9   7   5   3   1   1   3   5   7   9   11

VIS                                   X                   VRB
     11   9   7   5   3   1   1   3   5   7   9   11

SEQ                                               X       GLO
     11   9   7   5   3   1   1   3   5   7   9   11
```

What’s my point?

- It is not a priori obvious (in a cognitive sense) whether either a textual or a visual approach should in fact dominate
- That is we don’t know that
  - “A picture is worth 1000 words”

    nor that

  - textual programming is inherently superior

Synopsis

- This was a quick, high-level overview of two very large areas
  - Visual programming
  - Program visualization
- We haven’t covered many of the systems that exist
  - We haven’t covered any of the systems in detail

Key question

- What domains (of computation and of users) would especially benefit from visualization?
  - This requires, I believe, some understanding of learning styles, some empirical HCI studies, some understanding of the computational domain, etc.

Next week

- Domain-specific languages
  - How can we leverage particular domains in which we'll be doing a set of related computations?
    - CLP(R) is one example we've seen of a DSL