LIFT: Reusing Knowledge from Legacy Systems

Kellyton dos Santos Brito

Informatics Center - Federal University of Pernambuco
C.E.S.A.R. - Recife Center for Advanced Studies and Systems
Kellyton.brito@cesar.org.br

- **Software Reuse**
  - Initial ideas from McIlroy (1968)

Software reuse is the process of ***creating software systems from existing software*** *rather than building them from scratch* (Krueger 1992)

- Reusable Assets
  - Products, Processes, Knowledge ...

- Reuse Aspects
  - Processes, methods, environments, tools and non-technical aspects

One (of many) point is...

Knowledge reuse from legacy systems

- **Legacy Systems**
  - Well **Tested**, **stable**, low bugs and defects
  - A lot of **embedded knowledge**

- **Problems**
  - **Obsolete** technologies, languages, tools and processes
  - Non useful **documentation**
  - Degradation due to maintenance operations
  - Few specialized **people**

- **Directions**
  - **Reverse engineer** applications
  - Knowledge **Reuse**

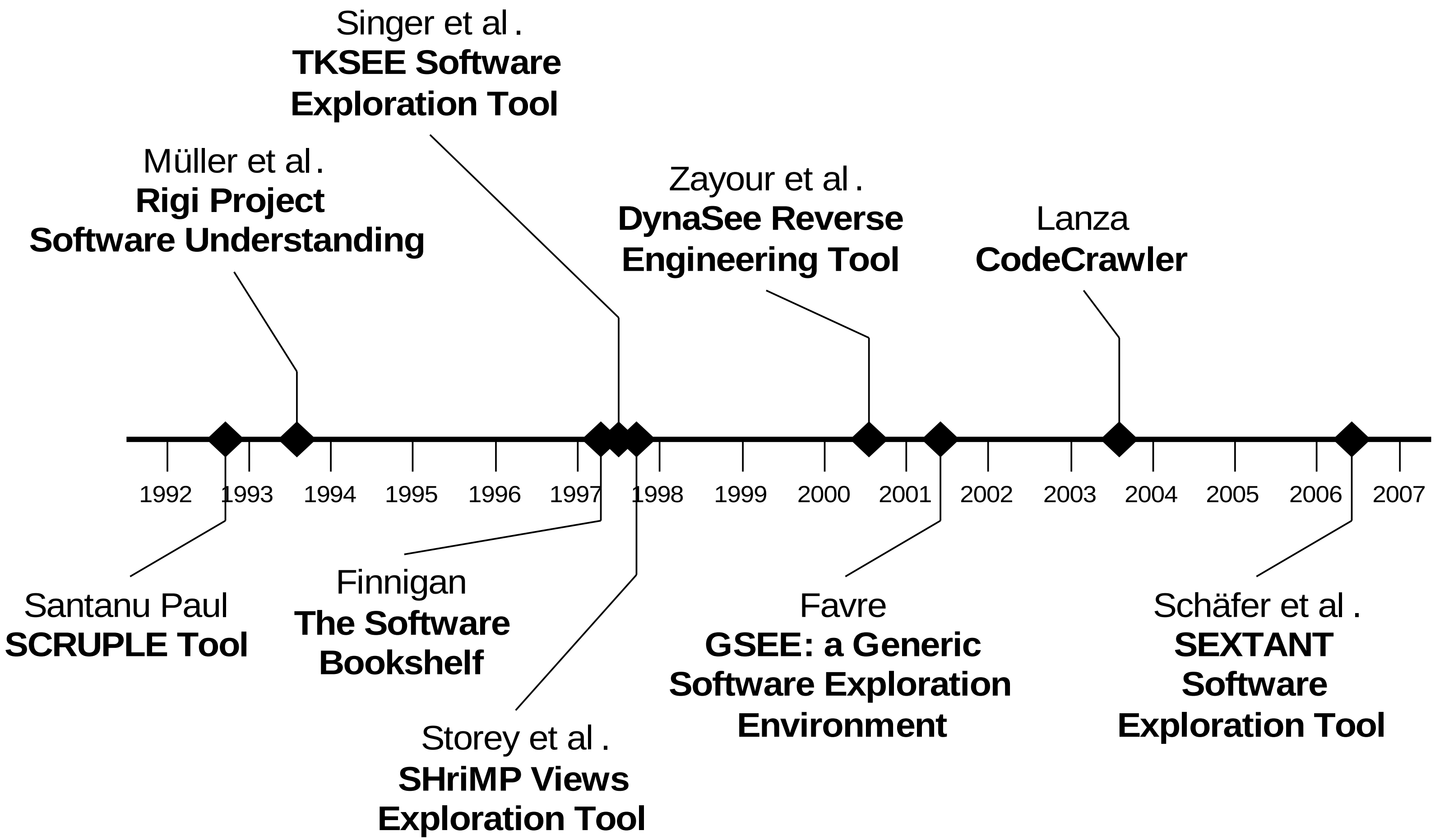
LIFT: Legacy Information Retrieval Tool

LIFT: Legacy InFormation retrieval Tool

- **Objective**
  - To automate tasks of reverse engineering and legacy systems knowledge reuse

- **The requirements**
  - Based on the state-of-the-art and practice in reengineering and reverse engineering

Reverse Engineering Tools



Reverse Engineering Tools

- Almost all of them shows a **call graph**
- Each one implements its **proper requirements set**
  - Some with **exploration** capabilities
  - Some with **visualization** capabilities
  - Some with **cognitive** capabilities
- All of them **highly user dependent**: lack of automatic or semi-automatic code analysis
- Lack of recover and **traceability** of entire system, **from interface to database**
- Discover **HOW** programs works, instead of **WHAT** programs do
- Problems dealing with **big systems**

LIFT Functional Requirements

(FR1) Visualization of entities and relations

(FR2) Abstraction mechanisms

(FR3) High user interactivity

(FR4) Search capabilities

(FR5) User activities trace capabilities

(FR6) Metrics visualization support

(FR7) Recovery of the entire system (interface, design and database)

(FR8) Trace of requirements from interface to database access

(FR9) Possibility of semi-automatic suggestions

Existent Requirements

New Requirements

LIFT Non Functional Requirements

(NFR1) Cross Artifacts support

(NFR2) Extensibility

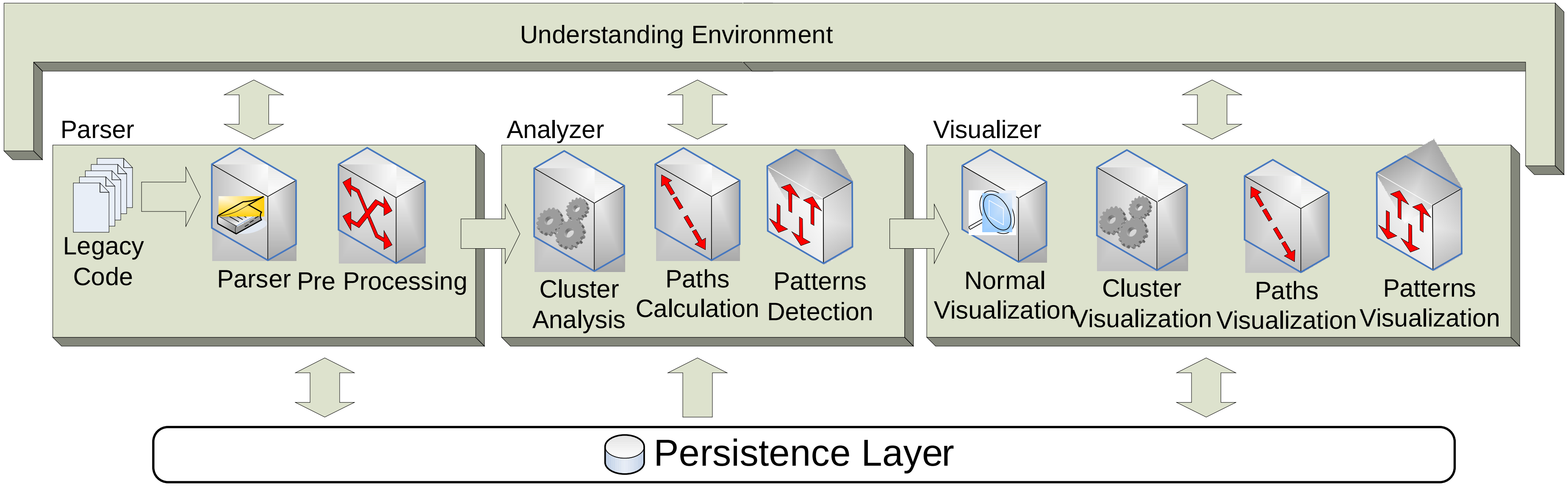
(NFR3) Integration with other tools

(NFR4) Scalability

(NFR5) Maintainability and Reusability

Existent Requirements
New Requirements

LIFT Architecture



Implementation: Parser Component

- **Parser Module**
  - Parses NATURAL/ADABAS source code
  - First version developed by *Pitang* team
  - Uses C# technology
  - Integrated as a component

- **Pre-Processing Module**
  - Works with parser output
  - Store useful information in the database
    - SQL ANSI
  - Performs the system slice
  - Deduction of database layer

Implementation: Analyzer Component

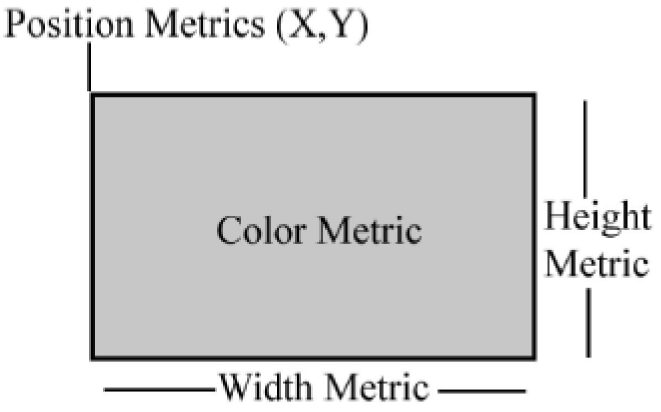
- Call Graph Generation
- Paths Calculations
  - Full paths
  - Minimal paths
    - Using Dijkstra shortest path algorithm
    - Running time $O(n.log\ n)$
- Cluster analysis
  - Hierarchical Clustering
  - Mark Newman's "*edge betweenness clustering algorithm*"
    - Running time $O(k.m.n)$
- Patterns detection (second interaction)
  - Text pattern detection
  - Graph pattern detection
  - Clone detection

Implementation: Visualizer Component

- Based on JUNG: Java Universal Network/Graph Framework

- Visualizations of call graph and Analyzer modules
  - Normal visualization
  - Cluster visualization
  - Paths visualization
  - Patterns visualization

- Uses *Polimetric-Views* concept



Understanding Environment

- Graphical interface
- Integrate the other components
- Shows source code
- Works with *views concept*
  - Isolate subgraphs
- Allow comments
  - Views comments
  - Modules comments
  - Source code comments

LIFT Usage: Initial Steps

- Parse
  - Ca

LIFT - Legacy I

File | System Under

Organize S

Visualizatio

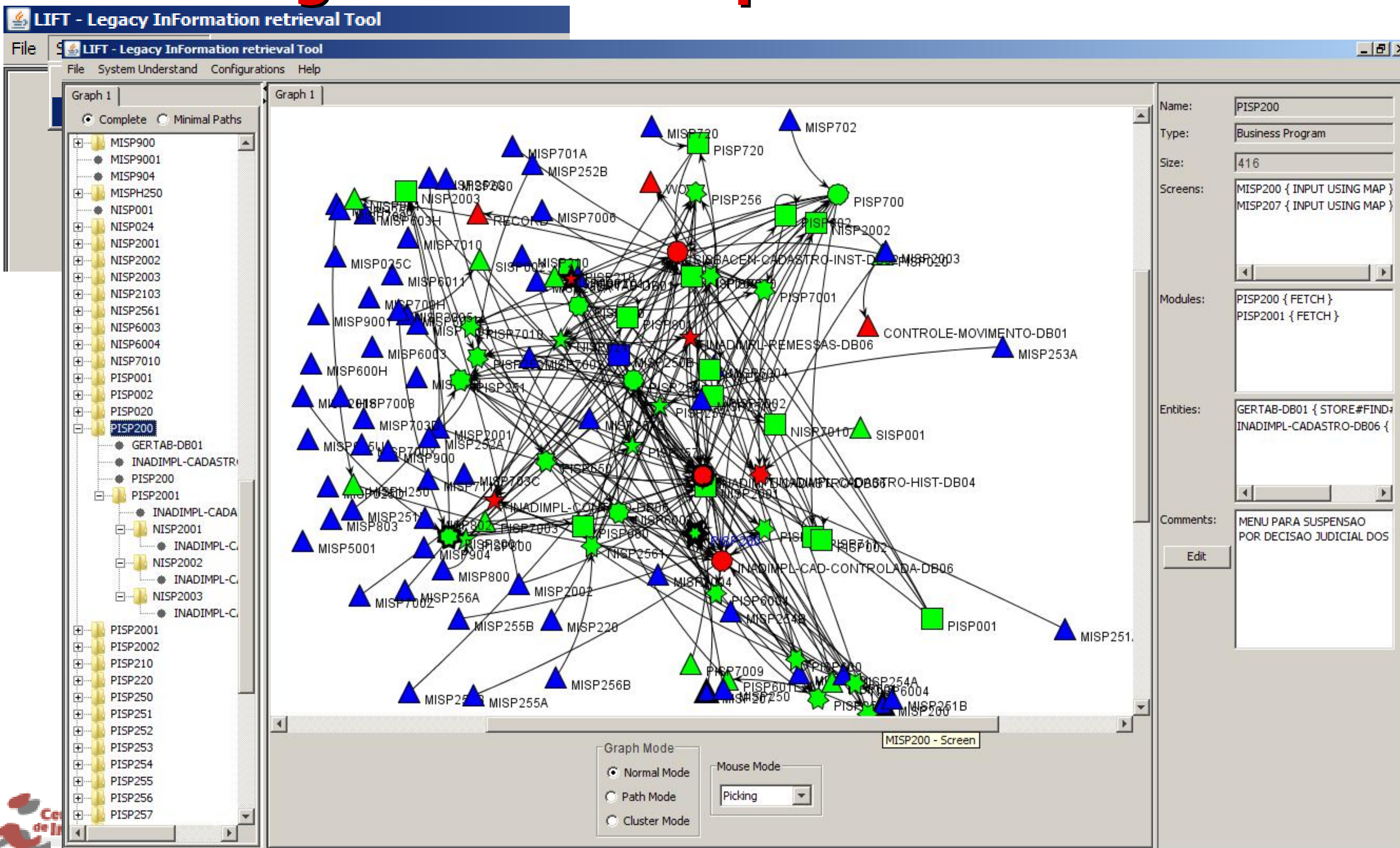
```
0010 *****************************************************************
0020 * PROGRAMM/TEXT:  SXRANDOM                USER/DATE: WIEGANDT / 20030513 *
0030 *
0040 *
0050 *
0060 *******
0070 DEFINE
0080 PARAMETI
0090 01 #RANI
0100 *
0110 01 #STAI
0120   02 #X
0130   02 #Y
0140   02 #Z
0150 *
0160 LOCAL
0170 01 #X-D
0180 01 #Y-D
0190 01 #Z-D
0200 *
0210 01 #T (
0220 01 #T-A
0230 *
0240 01 #GEN1
0250 01 #GEN2 (I4)
0260 01 #GEN3 (I4)
0270 01 #TEMP (I4)
0280 END-DEFINE
0290 *
0300 PERFORM RANDOM
0310 *
0320 *****************************************************************
0330 DEFINE SUBROUTINE RANDOM
0340 PERFORM GEN1
0350 PERFORM GEN2
0360 PERFORM GEN3
0370 COMPUTE #T = #GEN1 / 30269.0 + #GEN2  / 30307.0 + #GEN3 / 30323.0
0380 COMPUTE ROUNDED #T-ADD = #T *1
0390 COMPUTE #RANDOM = #T - #T-ADD
0400 END-SUBROUTINE /* RANDOM
0410 *****************************************************************
0420 DEFINE SUBROUTINE GEN1
0430 IF #X = 0 OR #X > 30000
```

Results | Messages

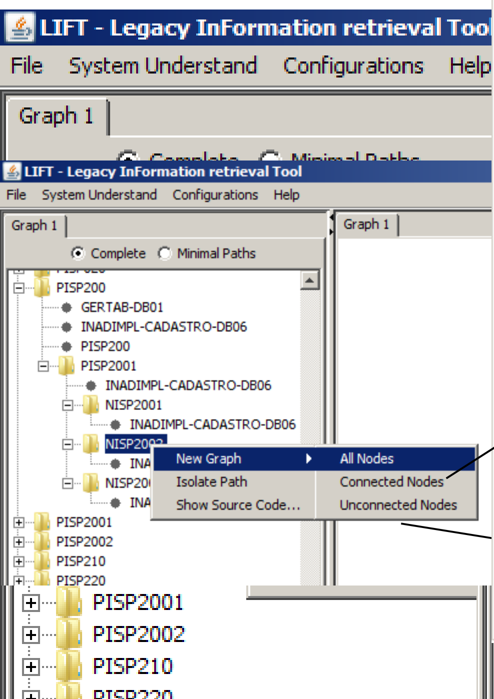
	sigla_demanda	sourcelinenumber	objectlinecontent	instruction	operand	result	statementlevel	subroutine	insideblock
25	RANDOM	26	01 #GEN3 (I4)	01			2		DEFINE DATA
26	RANDOM	27	01 #TEMP (I4)	01			2		DEFINE DATA
27	RANDOM	28	END-DEFINE	END-DEFINE			1		DEFINE DATA
28	RANDOM	29	*				1		
29	RANDOM	30	PERFORM RANDOM	PERFORM			1	RANDOM	
30	RANDOM	31	*				1		
31	RANDOM	32	**...				1		
32	RANDOM	33	DEFINE SUBROUTINE RANDOM	DEFINE SU...			1		DEFINE SU...
33	RANDOM	34	PERFORM GEN1	PERFORM			2	GEN1	DEFINE SU...
34	RANDOM	35	PERFORM GEN2	PERFORM			2	GEN2	DEFINE SU...
35	RANDOM	36	PERFORM GEN3	PERFORM			2	GEN3	DEFINE SU...
36	RANDOM	37	COMPUTE #T = #GEN1 / 30269.0 + #GEN2 / 30307.0 ...	COMPUTE	#GEN1 / 30269.0...	#T	2		DEFINE SU...
37	RANDOM	38	COMPUTE ROUNDED #T-ADD = #T *1	COMPUTE	#T *1	ROUNDED #T-ADD	2		DEFINE SU...
38	RANDOM	39	COMPUTE #RANDOM = #T - #T-ADD	COMPUTE	#T - #T-ADD	#RANDOM	2		DEFINE SU...
39	RANDOM	40	END-SUBROUTINE /* RANDOM	END-SUBR...			1		DEFINE SU...

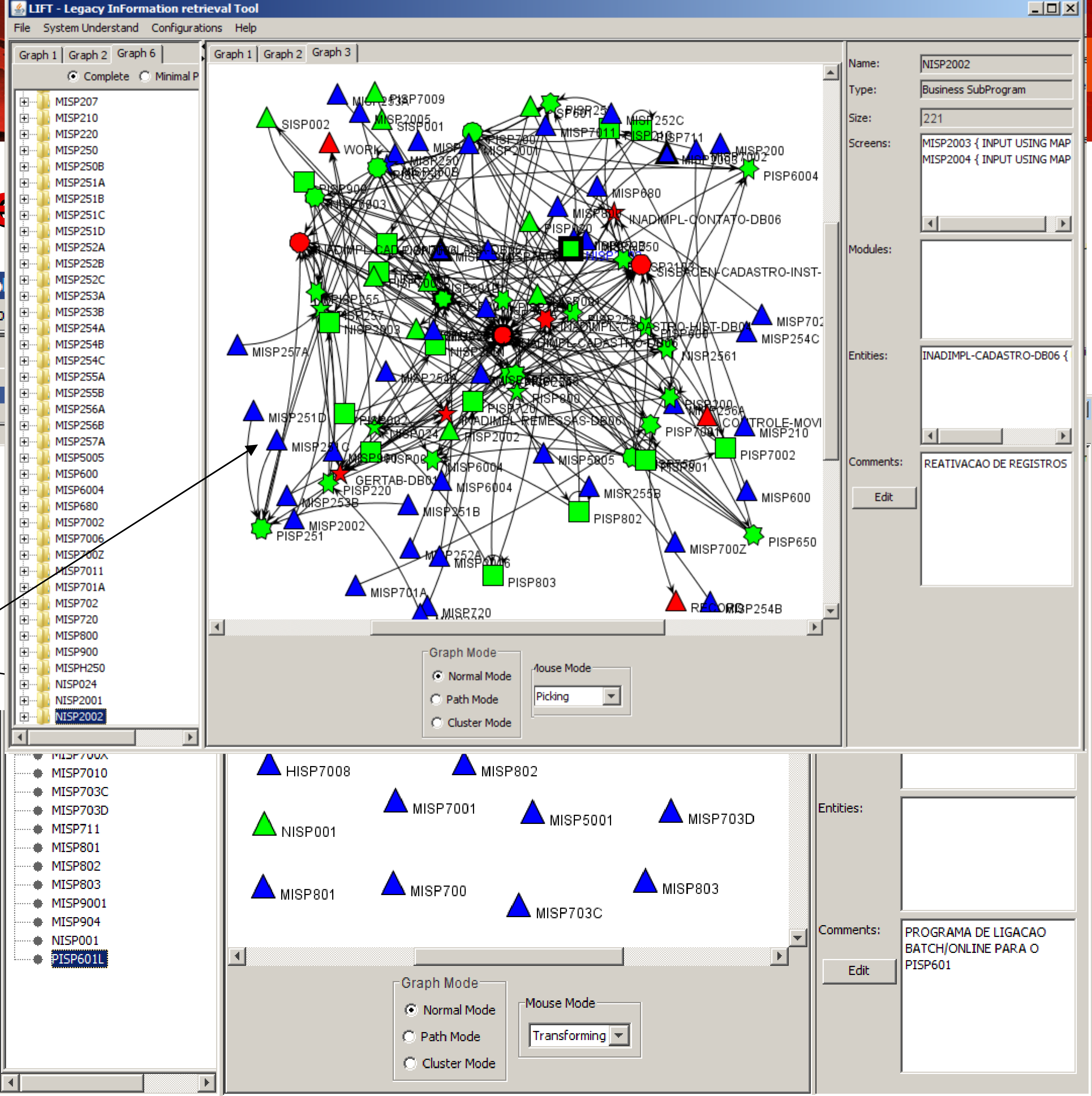
Query executed successfully. | KBRIPC\SQLEXPRESS (9.0 SP1) | kellyton (52) | Legacy2ooDB | 00:00:00 | 110 rows

LIFT Usage: Initial Graph

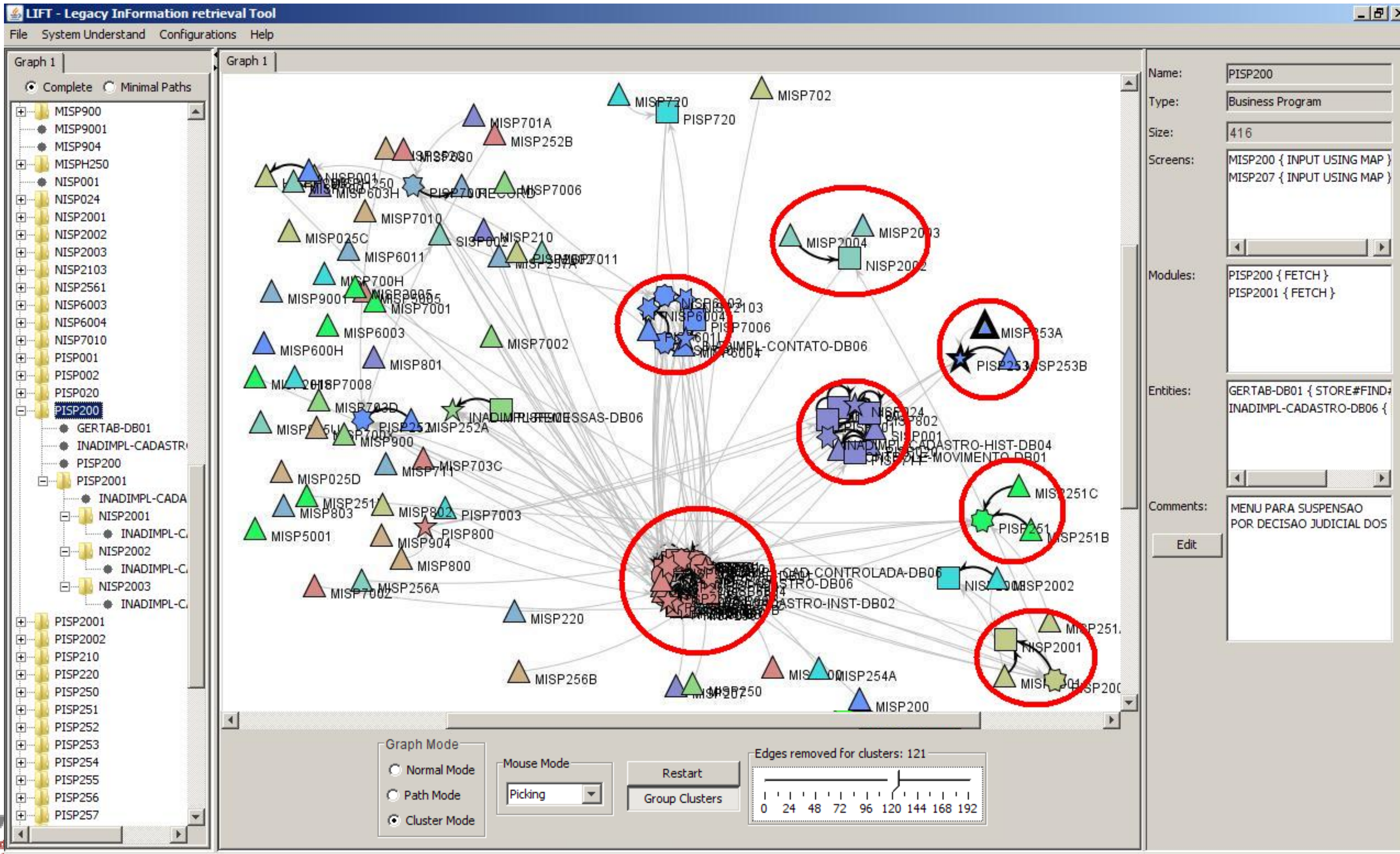


LIFT Usage


LIFT - Legacy InFormation retrieval Tool
File
System Understand
Configurations
Help
Graph 1
Complete
Minimal Paths
PISP200
GERTAB-DB01
INADIMPL-CADASTRO-DB06
PISP2001
NISP2001
NISP2002
New Graph
Isolate Path
Show Source Code...
All Nodes
Connected Nodes
Unconnected Nodes
PISP2002
PISP210
PISP220

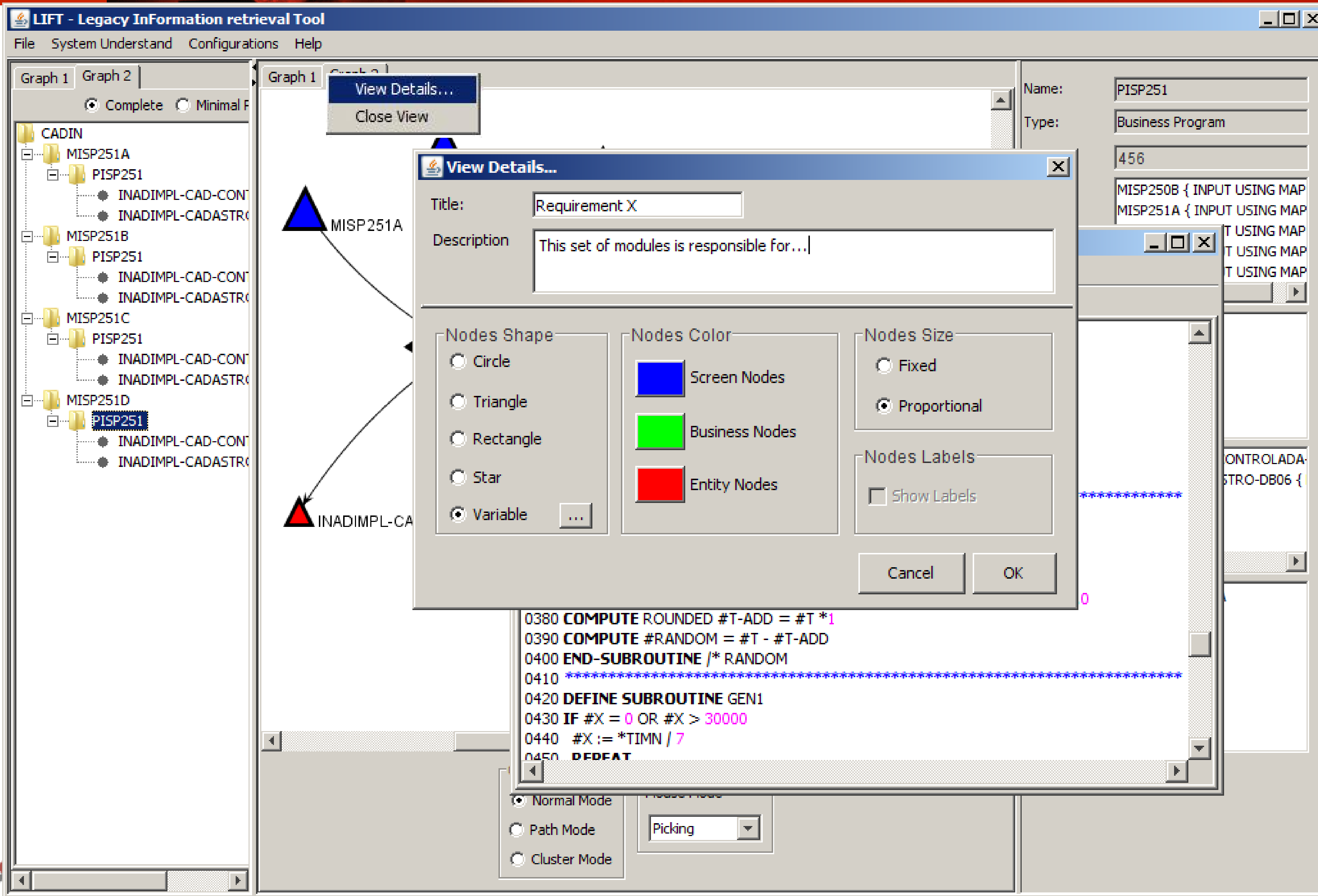

LIFT - Legacy InFormation retrieval Tool
File
System Understand
Configurations
Help
Graph 1
Graph 2
Graph 6
Graph 3
Complete
Name:
NISP2002
Type:
Business SubProgram
Size:
221
Screens:
MISP2003 { INPUT USING MAP
MISP2004 { INPUT USING MAP
Modules:
Entities:
INADIMPL-CADASTRO-DB06 {
Comments:
REATIVACAO DE REGISTROS
Edit
Graph Mode
Normal Mode
Path Mode
Cluster Mode
Mouse Mode
Picking
Transforming
PROGRAMA DE LIGACAO BATCH/ONLINE PARA O PISP601

LIFT Usage: Detecting Clusters



R:SE

REUSE IN SOFTWARE ENGINEERING GROUP



Case Study

The Context

- **Pitang Software Factory**
  - Infra-structure
  - Experienced staff
  - Real demands for reverse engineering
    - NATURAL/ADABAS systems of a financial institution
    - Previous experience with reverse engineering: Almost 2 million LOC

Questions

- Does the tool provides **effort reduction** in reverse engineering projects?
- Does the tool **is scalable** to be used in large projects?
- Do the subjects have **difficulties** to use the tool?

The Planning

- **Method of comparison**
  - Comparison with two sibling projects
    - Same **technologies**: NATURAL/ADABAS
    - Same **domain**: Financial
    - Same **customer**
    - Same understanding **process**
    - Same number of **participants**
    - Similar engineers **experience**: more than 10 years

    - ***Different tools***

- **The Projects**
- LIFT Project: 210 KLOC system
- Sibling projects: 65 KLOC and 131KLOC systems

The Quantitative Analysis

Variable	Project 1	Project 2	LIFT Project
Lines of Code (LOC)	64929	131285	207689
Number of Modules	142	119	304
Understanding Effort (hours)	120	206	231
Productivity: lines/hour	541,08	637,31	899,09
Productivity: modules/hour	1,18	0,58	1,32

- **Lines/Hour Productivity**
  - 66% higher than *Project 1* and 41% higher than *Project 2*

The Quantitative Analysis

Variable	Project 1	Project 2	LIFT Project
Lines of Code (LOC)	64929	131285	207689
Number of Modules	142	119	304
Understanding Effort (hours)	120	206	231
Productivity: lines/hour	541,08	637,31	899,09
Productivity: modules/hour	1,18	0,58	1,32

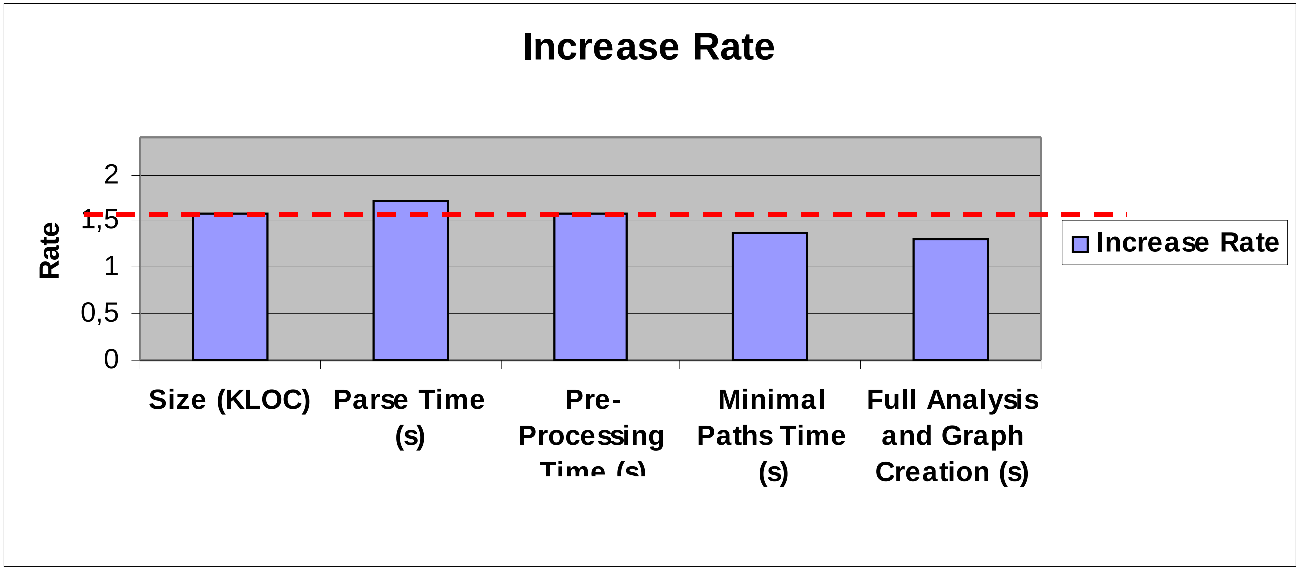
- **Modules/Hour Productivity**
  - 12% higher than *Project 1* and 127% higher than *Project 2*

The Quantitative Analysis

- Scalability
  - LIFT project and "Project 2" evaluation

Pentium IV / 512MB Database Server x Dual Core 2 / 2GB Client

Project	Project 2	LIFT Project
Size (KLOC)	131,285	207,689
Parse Time (s)	364	621
Pre-Processing Time (s)	93	146
Minimal Paths Time (s)	24	33
Full Analysis and Graph Creation (s)	30	39



The Qualitative Analysis

- Based on a questionnaire
- Tool effectivity
  - Effort reduction of about 20%
  - Easy to locate system features and to generate system documentation
- Weak Point
  - Delay to load the application (*Full Analysis and Graph Creation*)

Case Study Summary

Questions

- Does the tool provides **effort reduction** in reverse engineering projects?
  - **Yes**
- Does the tool **is scalable** to be used in large projects?
  - **Yes**
- Do the subjects have **difficulties** to use the tool?
  - **No**

LIFT: Reusing Knowledge from Legacy Systems

Kellyton dos Santos Brito

Informatics Center - Federal University of Pernambuco
C.E.S.A.R. - Recife Center for Advanced Studies and Systems
Kellyton.brito@cesar.org.br