An Integrated Approach for Studying Architectural Evolution

Qiang Tu and Michael Godfrey
Software Architecture Group (SWAG)
University of Waterloo

Overview

- Challenges in studying software evolution
- Motivation of our approaches
- "Origin analysis" and BEAGLE tools
- Case study – from GCC to EGCS

Challenges in Studying Software Evolution

Challenge 1: Modeling and Analysis

- How to model/measure changes
  - "Additive" and "Invasive"
- What is the implication of changes

Challenge 2: Tool Support

- Visualization and navigation
- Integrated environment

Challenge 3: Data Management

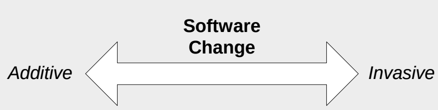
- What data are relevant
- How to efficiently store and query data

Motivation

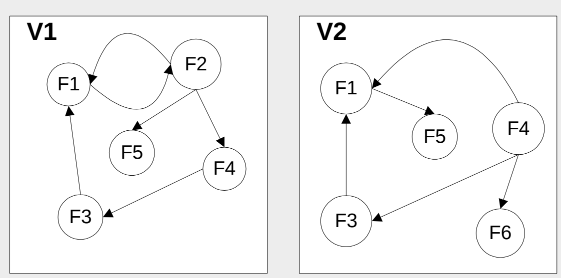
- **Entity – Relation – Version** data model
  - Based on source code and reverse engineering
  - Entity and Relation
    - Extracted and "lifted" architecture facts
    - Atomic and composite entities
  - Release
    - Extract facts for every release of the software system
    - Add a "release" column to [entity a, entity b, relation] tuple
  - Store in relational database
  - Query with SQL statements

Motivation (cont.)

- Evolution model for "**invasive**" changes



- <u>Additive</u> changes
  - Daily development activities
  - Adding, removing and modifying -
    - Code lines / Functions / Files / Subsystems
  - Assume a change in name/location of a entity means the old is out and a new is in
  - Study with ***diff*** and ***relational calculus***



New entities: F6

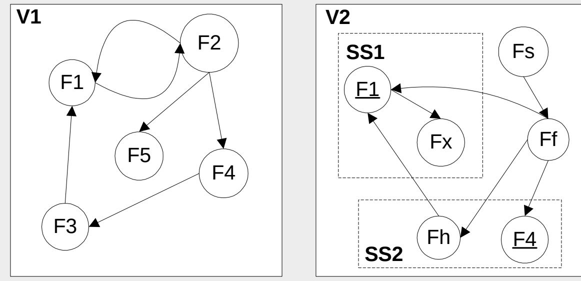
Deleted entities: F2

Changed entitles: *diff* on pairs with same function name

Changed relations: *grok or SQL*

Motivation (cont.)

- Invasive changes
  - Structural and architectural changes
  - Results of :
    - Refactoring / code cleaning
    - Redesign of the system
  - Break old name/location model
  - Difficulties:
    - How to define an entity to be ***new***?
    - How to measure the difference between the different versions of the same entity?



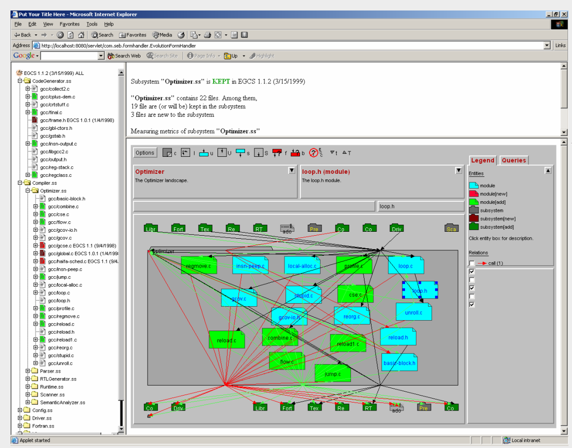
Possible solutions:

- *match “fingerprints”*
- *relations with stable entities*

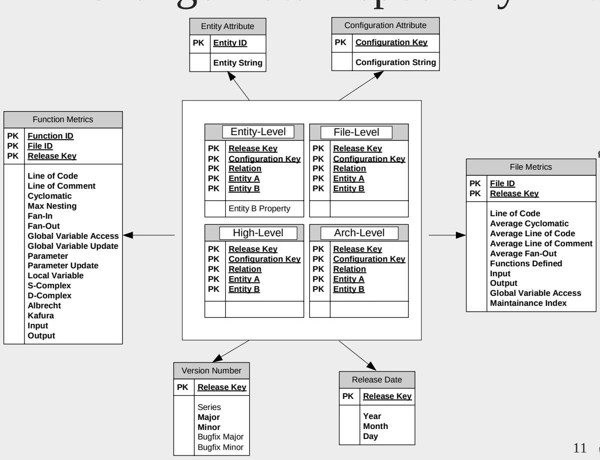
Motivation (cont.)

- Build a set of tools and integrated environment
  - Aid in understanding how software evolves
  - Compare the architecture of multiple releases
    - Additive
    - Invasive
  - Visualize and navigation tools
  - Analyze the meanings of changes

Beagle Environment



Change Data Repository



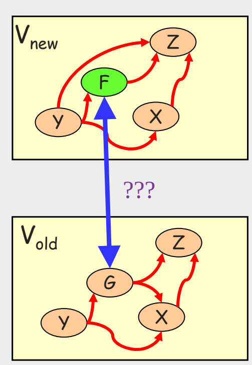
“Origin Analysis”

Suppose that:

- **F** is the name of a software entity (e.g., function, type, global variable) of version V_{new} of a software system.
- There is no entity of the same name/kind in the previous version V_{old}

We define origin analysis as the process of deciding:

- if **F** was newly introduced in V_{new}, or
- if it should be more accurately viewed as a changed/moved/ renamed version of a differently named entity of V_{old}



Origin analysis: Two techniques

Entity analysis (i.e., metrics-based Bertillonage)

- For each “new” entity **f**:
  - Calculate combined Euclidean distance from each “deleted” entity for five metrics:

    (S-Complexity, D-Complexity, Cyclomatic, Albrecht, Kafura)

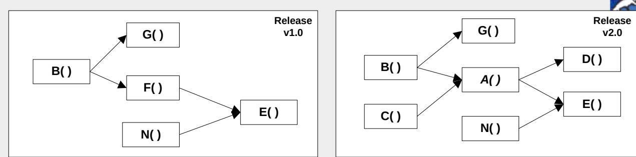
    [Kontogiannis]
  - Select top **k** matches; compare entity names.

13

Origin analysis: Two techniques

Relationship analysis (e.g., calls, data refs)

- For each “new” entity **f**:
  - Find R_f, set of all entities that call **f** that are present in both versions.
  - For each $g \in R_f$, calculate Q_g, set of all “deleted” entities that **g** calls in the old version.
  - Look at intersection of the Q_gs; these are good candidates.



Efficiency considerations

- When comparing V_{new} to V_{old}, need to find the entities that seem to have been added and deleted.
  - These sets are fast to determine.
  - Most subsequent calculations involve only these small subsets of the entire entity space.
- Computationally expensive approaches for clone detection (e.g., graph matching) were not considered.
  - Can’t pre-compute easily.
  - Precise matching not worth the effort, as it doesn’t seem to help much for this task.

15

Efficiency considerations

- Entity analysis:
  - Entity info is generated by fact extractor and metrics tool.
    - Info is generated only once per version, when system is checked into repository.
  - Performing entity analysis is a matter of a simple numerical calculation on a small set of “likely candidates”.
- Relationship analysis:
  - Relationship info (who-calls-whom, who-inherits-from-whom, etc.) is generated by fact extractor.
    - Info is generated only once per version, when system is checked into repository.
  - Computation and comparison of relational images is fairly fast.
    - Special-purpose tool (`grok`) and relatively small amount of data.

16

Usage of BEAGLE

At system check-in:

- Populate database with “facts” and metrics info from various tools.
- `grok` scripts “lift” facts to file/ subsystem /architectural level.

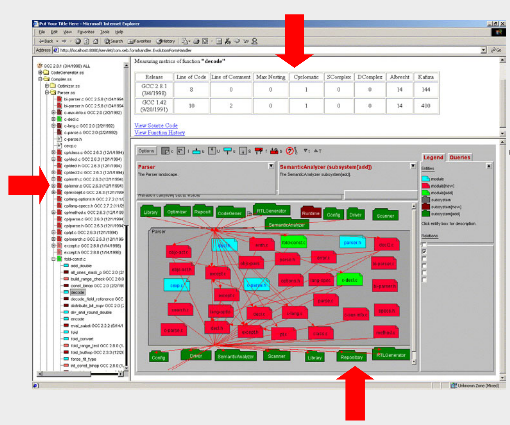
At runtime:

- PBS engine for visualization/navigation.
- Java-based infrastructure using DB/2, VA-Java, IBM-Websphere.

17

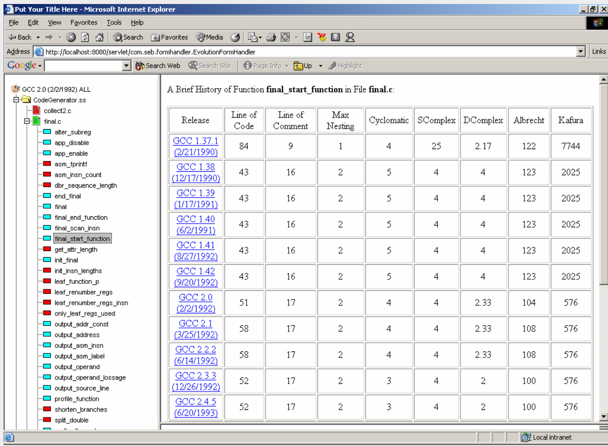
Metric history for selected entities

Overview of system structure changes



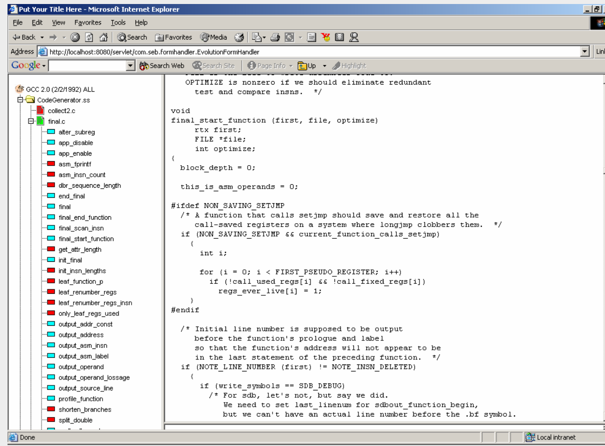
Visualize the diff between two versions

18



A Brief History of Function **final_start_function** in File **final.c**:

Release	Line of Code	Line of Comment	Max Nesting	Cyclomatic	SComplex	DComplex	Albrecht	Kafura
GCC 1.37.1 (2/21/1990)	84	9	1	4	25	2.17	122	7744
GCC 1.38 (12/17/1990)	43	16	2	5	4	4	123	2025
GCC 1.39 (1/17/1991)	43	16	2	5	4	4	123	2025
GCC 1.40 (6/2/1991)	43	16	2	5	4	4	123	2025
GCC 1.41 (8/27/1992)	43	16	2	5	4	4	123	2025
GCC 1.42 (9/20/1992)	43	16	2	5	4	4	123	2025
GCC 2.0 (2/2/1992)	51	17	2	4	4	2.33	104	576
GCC 2.1 (3/25/1992)	58	17	2	4	4	2.33	108	576
GCC 2.2.2 (6/14/1992)	58	17	2	4	4	2.33	108	576
GCC 2.3.3 (12/26/1992)	52	17	2	3	4	2	100	576
GCC 2.4.5 (6/20/1993)	52	17	2	3	4	2	100	576

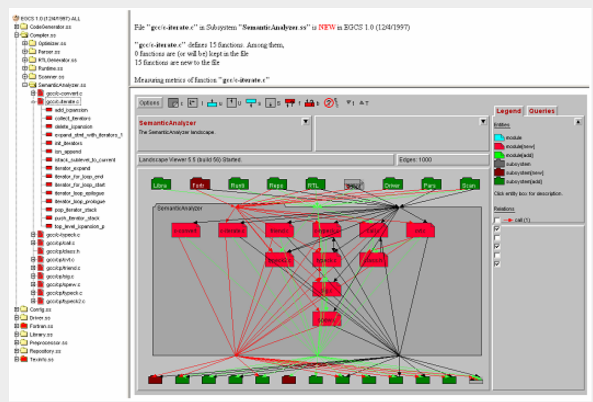


```
   OPTIMIZE is nonzero if we should eliminate redundant
     test and compare insns.  */

void
final_start_function (first, file, optimize)
     rtx first;
     FILE *file;
     int optimize;
{
  block_depth = 0;

  this_is_asm_operands = 0;

#ifdef NON_SAVING_SETJMP
  /* A function that calls setjmp should save and restore all the
     call-saved registers on a system where longjmp clobbers them.  */
  if (NON_SAVING_SETJMP && current_function_calls_setjmp)
    {
      int i;

      for (i = 0; i < FIRST_PSEUDO_REGISTER; i++)
        if (!call_used_regs[i] && !call_fixed_regs[i])
          regs_ever_live[i] = 1;
    }
#endif

  /* Initial line number is supposed to be output
     before the function's prologue and label
     so that the function's address will not appear to be
     in the last statement of the preceding function.  */
  if (NOTE_LINE_NUMBER (first) != NOTE_INSN_DELETED)
    {
      if (write_symbols == SDB_DEBUG)
        /* For sdb, let's not, but say we did.
           We need to set last_linenum for sdbout_function_begin,
           but we can't have an actual line number before the .bf symbol.
```


Case study: `gcc/g++/egcs`

- Have extracted full info for 29 versions of `gcc/g++/egcs`
  - Want to examine major breaks in development to see how well origin analysis works.
- EGCS v1.0 was forked from the GCC v2.7.2.3 codebase
  - EGCS project goals:
    - C++ compiler more ANSI compliant,
    - new FORTRAN front-end,
    - new optimizations and code-generation algorithms, …
  - … and EGCS introduced a new directory structure and a new file naming scheme, in addition to all of the other redesign and restructuring.
  - Naïve analysis indicated "everything old is new again" ☹

Case study: `gcc/g++/egcs`



Case study: `gcc/g++/egcs`

- Example:
  - The EGCS 1.0 **`Parser`** subsystem contains 15 (non-trivial) implementation files, comprising 848 functions.
  - Using origin analysis and common sense, we decided that about half of the "new" functions weren't new.
  - That's still a massive amount of change for a new release of a compiler!

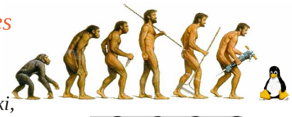
File	# Fcns	# New	# Old	% New
gcc/cp/errfn.c	9	9	0	100%
gcc/cp/pt.c	59	57	2	97%
gcc/except.c	55	52	3	95%
gcc/cp/decl2.c	57	50	7	88%
gcc/c-lang.c	16	14	2	88%
gcc/cp/method.c	30	26	4	87%
gcc/cp/except.c	25	20	5	80%
gcc/cp/decl.c	134	84	50	63%
gcc/cp/error.c	31	16	15	52%
gcc/cp/class.c	61	31	30	51%
gcc/cp/search.c	81	40	41	49%
gcc/c-decl.c	70	29	41	41%
gcc/fold-const.c	44	15	29	34%
gcc/objc/objc-act.c	167	17	150	10%
gcc/c-aux-info.c	9	0	9	0%
TOTAL	848	460	388	54%

Conclusion and Open Questions

- Beagle: An Integrated Platform
  - What are other models for additive and invasive changes?
  - Requires more case studies and validation.
- Origin Analysis
  - Requires human intervention to make intelligent decisions.
  - Techniques need to be fast and approximate. We need more of them.

IWPSE-03

- *2003 Intl. Workshop on Principles of Software Evolution*
  - *To be held Sept 1-2, 2003 in Helsinki, Finland*
    - *Co-located with FSE/ESEC 2003*
    - *CFP to appear in early 2003*
  - *General chair:*
    - *Tommi Mikkonen*
  - *Program co-chairs:*
    - *Motoshi Saeki*
    - *Mike Godfrey*



esec fse 2003