Motivation

- Underlying question: *How does software change?*
  - In: Two versions of a program
  - Out: Picture of changes

- Relevance
  - Software development
  - Software engineering
Objective and Approach

- Summarize C program changes
  - Functions (body AST, prototype)
  - Global variables (type and initializer)
  - Types
    - Structs/Unions (fields deleted / added / type changed)
    - Typedefs
    - Enums

- Our Approach: AST matching
  - Accurate; handles renamings
  - Scales to real-world applications; e.g., Apache, Linux kernel, OpenSSH
struct "net_device": 1 fields changed type: “accept_fastpath”
struct "reiserfs_journal": 1 fields deleted: “j_dummy_inode”
struct "reiserfs_journal": 1 fields added: “j_dirty_buffers”

function "block_read_full_page": 1 arguments changed type: “get_block”
function "ext2_readdir": 1 arguments changed type: “filldir___0”

+ function “inetdev_changename”
+ function “__ide_dma_good_drive”
+ function “ide_unplugged_outbsync”
+ function “inode_init_once”
- function “target_cpus”
- function “ide_dmafunc_verbose”

+ typedef “cisco_proto”
- typedef “ide_ioctl_proc”

+ global var “idecd”

Linux 2.4.20 vs 2.4.21
The Renaming Problem

Same program, syntactic changes only

```c
typedef int sz_t;

struct foo {
    int i;
};

int count;

void f(int a) {
    struct foo sf;
    sz_t c = 2;
    sf.i = a + c;
    count++;
}

Version 1

typedef int size_t;

struct bar {
    int i;
};

int counter;

void f(int b) {
    struct bar sb;
    size_t d = 2;
    sb.i = b + d;
    counter++;
}

Version 2
```
Abstract Syntax Tree Matching

Compare ASTs for functions with same name

Program Version 1 → Parsing → AST 1

Program Version 2 → Parsing → AST 2

AST Matching

AST Traversal Map Generation

Renaming Detection

AST Traversal Change Detection

Changes & Statistics

Understanding Source Code Evolution Using Abstract Syntax Tree Matching
Understanding Source Code Evolution Using Abstract Syntax Tree Matching

void f(int a) {
    struct foo sf;
    sz_t c = 2;
    sf.i = a + c;
    count++;
}

Version 1

void f(int b) {
    struct bar sb;
    size_t d = 2;
    sf.i = b + d;
    counter++;
}

Version 2

Name Map

AST Traversal - Name Map Generation
Understanding Source Code Evolution
Using Abstract Syntax Tree Matching

void f(int a) {
    struct foo sf;
    sz_t c = 2;
    sf.i = a + c;
    count++;
}

Version 1

void f(int b) {
    struct bar sb;
    size_t d = 2;
    sf.i = b + d;
    counter++;
}

Version 2
Abstract Syntax Tree Matching

Traverse the ASTs in parallel, computing changes

AST Matching

Program Version 1 → Parsing → AST 1
Program Version 2 → Parsing → AST 2

AST Traversal Map Generation → Renaming Detection → AST Traversal Change Detection

Changes & Statistics

A renamed to B iff
• A → B in the map
• A deleted
• B added

Understanding Source Code Evolution Using Abstract Syntax Tree Matching
typedef int sz_t;
struct foo {
    int i;
    sz_t f;
}  

typedef int size_t;
struct foo {
    long long i;
    size_t f;
    double e;
}  

Version 1  Version 2

Field i changed type: int -> long long
Field e added

STOP
Implementation

- Parsing via CIL toolkit
  - Merges whole program into single, preprocessed file
- Fast
  - Scales linearly, 400,000 LOC in 1 minute
- Generates different output formats
  - Raw differences, summaries, density trees
Summary Statistics

------- Functions -------
Version1 : 7697
Version2 : 7881
  added : 232
  deleted : 48
  locals/formals changed name : 3
  arguments type changes : 19
  return types changes : 15

------- Structs/Unions -------
Version1 : 1214
Version2 : 1233
  added : 17
  deleted : 1
  field type changes : 15
  field count changes : 19

------- Typedefs -------
Version1 : 487
Version2 : 469
  added : 13
  deleted : 31
  base type changes : 2

------- Global Variables -------
Version1 : 8027
Version2 : 8074
  added : 43
  deleted : 16
  var type changes : 11
  var val changes : 51

------- Enums -------
Version1 : 33
Version2 : 31
  deleted : 2
  item count changes : 1
  var exp changes : 20

Linux 2.4.20 vs 2.4.21
Density Trees

Struct/Union field additions Linux 2.4.20 vs 2.4.21

Understanding Source Code Evolution Using Abstract Syntax Tree Matching
Case Studies: OpenSSH, Vsftpd, Apache

Functions & global variables: how often added and deleted?

- OpenSSH changes most frequently
- Deletions infrequent, relative to additions
Case Studies: OpenSSH, Vsftpd, Apache

How often do function bodies and prototypes change?

- Function bodies do change a lot
- Function prototypes do not change much
Related Approaches

- **Standard diff**
  - Low-level
  - Verbose: Linux 2.4.20->2.4.21 patch: 21MB

- **Release notes**
  - High level
  - Possibly incomplete
Summary

- Approach for reporting changes to C programs
  - AST-matching
  - Variety of changes at several levels of detail
  - Accurate
  - Scalable

- Soon to be available at