Evolution, Growth, and Cloning in Linux: A Case Study

University of Waterloo

Michael W. Godfrey
Davor Svetinovic
Qiang Tu

University of Waterloo

Consortium for Software Engineering Research
Consortium de recherche en génie logiciel

Overview

- Ongoing CSER project:
  - Investigating growth and evolution of open source software
    - Linux, vim, gcc, …
- Lehman's laws of evolution and Linux
  - Why is Linux still growing so fast?
    - Hyp: cloning is common
- Case study of Linux SCSI drivers (in progress)
  - How/why does cloning really occur?
  - Parallel evolution?
  - How well do clone detection tools work in spotting "real-world" cloning?

What is software evolution?

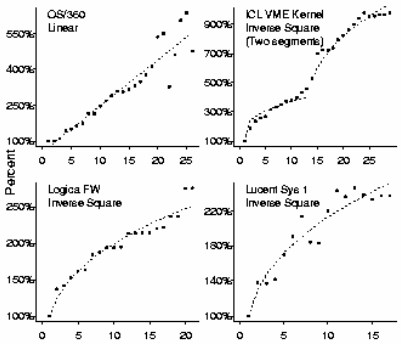
"Evolution is what happens while you're busy making other plans."

- Usually, we consider evolution to begin once the first version has been delivered:
  - *Maintenance* is the planned set of tasks to effect changes.
    - e.g., corrective, perfective, adaptive, preventive
  - *Evolution* is what actually happens to the software.

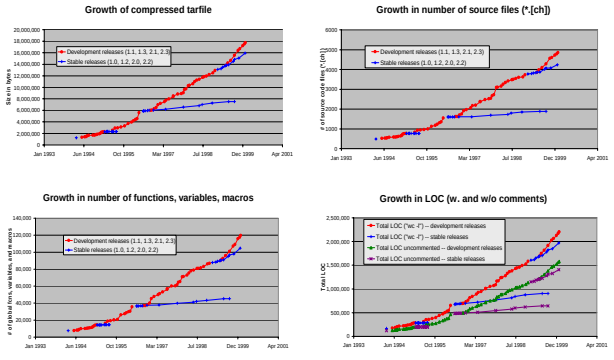
Lehman's Laws of software evolution in a nutshell

- Observations:
  - (Most) useful software must evolve or die.
  - As a software system gets bigger, its resulting complexity tends to limit its ability to grow.
  - Development progress/effort is (more or less) constant.
- Advice:
  - Need to manage complexity.
  - Do periodic redesigns.
  - Treat software and its development process as a feedback system (and not as a passive theorem).

Lehman's examples



Growth of Linux



Observations and hypotheses

- Growth along devel. path is super-linear

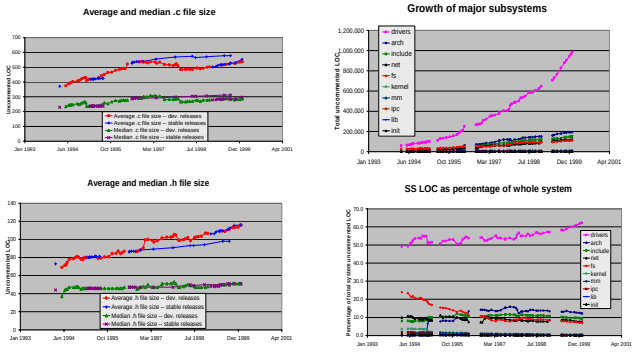
  y = .21*x^2 + 252*x + 90,055 r2=.997

  y = size in LOC
  x = days since v1.0
  r2 is "coefficient of determination" using least squares

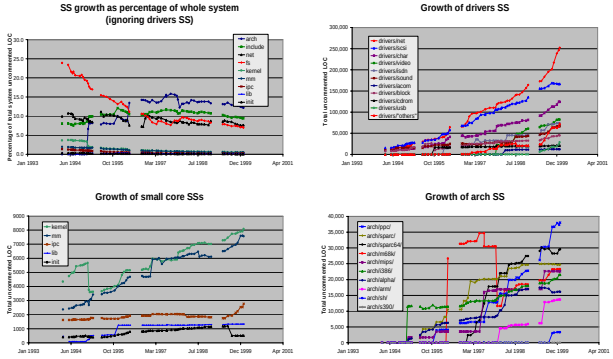
  [Lehman/Turski's model: y' = y + E/y^2 ✠ (3Ex)^(1/3)]

  - Linux's strong growth is continuing.
  - This is stronger growth at MLOC level than observed by others (Lehman, Gall), even for other OSs.

Linux growth phenomena



Linux growth phenomena



Why has Linux been able to continue its geometric growth?

- Core code quality is carefully maintained
- Architecture/problem domain
  - It's largely drivers
  - Much of the code is "parallel"
  - It's not as big as you might think
    - Vanilla configuration used only 15% of files
- Development model (OSD) and its sociology
  - Popularity and visibility has encouraged outsiders (both hackers and industry) to contribute
  - "Clone and hack" is an acceptable development style

Case study: Linux SCSI drivers

- Nice, controlled experiment:
  - Large body of code, multiple versions, well used system, open source
  - SCSI drivers all do similar tasks
  - Source comments shows cloning has occurred!
- Approx. 500 releases of Linux since 1994.
- Kernel v2.3.39: (released Jan 2000)
  - 5000 source files, 2.2 MLOC, 10 hardware architectures
  - `drivers/scsi` has 212 source files, 166 KLOC,

Goals of case study

- Examine "real world" cloning:
  - How common is it?
  - Why is it done?
  - What do the "cloning patterns" look like?
- Examine parallel evolution:
  - What kinds of changes are common?
  - Do developers (need to) change clone relatives too?
- Is there a better design structure lurking?
- Compare against clone detection tools
  - Are detections tools looking for the right indications of cloning?

SCSI Subsystem - Size (rel. 2.2.16)

- Number of source files: 211
- Number of functions: 2512
- Number of lines: 254,953
- % of comments: 38
- Number of low-level drivers: 80
- File size:
  - on average ~3000 lines
  - large multi-card drivers ~15,000 lines

SCSI Subsystem - Architecture

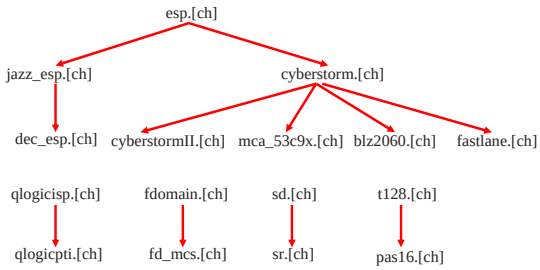
- Upper Layer
  - Uniform way of handling devices
  - Hard Disk, CD-ROM Disk, Tape, Generic
- Middle Layer
  - "bridge" between Upper Layer and Low-Level Devices
- Low-Level Device Drivers
  - low-level driver functionality and management

Clones Expected?

- Why did we expect to find clones:
  - Every driver must implement uniform interface
  - Design of subsystem does not support other forms of reuse
  - Driver logic is relatively simple (!)
  - Devices from same family ⇒ more cloning
  - Completely different hardware ⇒ less or no cloning
  - Open source ⇒ anyone can reuse code
  - Easier and more efficient to reuse existing code
  - Reused code already tested, so probably better quality than if we build it from scratch

Clones - Manual Inspection

- From source code comments, we have found:



Types of Changes Detected

- Names of variables
- Initialization parameters and constants
- Driver specific initialization logic removed/added
- Small change in supporting functions
- Small changes in driver management code
- Comments are updated
- Code changed is highly embedded into other code, which makes extraction of that code hard

Automatic Clone Detection

- We have looked for commercial and research clone detection software
- Clone Finder - www. studio501.com
  - free trial edition (C, C++)
  - easy to use
  - groups clones and highlights them in the source code
- Clone DR [Baxter] www.semdesigns.com (future)
  - Cobol trial edition (supports also C, C++, Java)
- Merlo et al. tool (future)

Clone Finder Results

- Number of files scanned: 8
- Number of source lines: 4081
- Elapsed time in seconds: 0.44
- Number of Groupings: 14
- Number of Blocks within those groupings: 30
- Total number of duplicated lines: 373
- Percent of source lines which are duplicated: 9.14

Something missed?

cyberstorm.c

```
....
static void dma_dump_state(struct NCR_ESP *esp)
{
        ESPLOG(("esp%d: dma -- cond_reg<%02x>\n",
                esp->esp_id, ((struct cyber_dma_registers *)
                                (esp->dregs))->cond_reg));
        ESPLOG(("intreq:<%04x>, intena:<%04x>\n",
                custom.intreqr, custom.intenar));
}

static void dma_init_read(struct NCR_ESP *esp, __u32 addr, int
        length)
{
        struct cyber_dma_registers *dregs =
                (struct cyber_dma_registers *) esp->dregs;

        cache_clear(addr, length);

        addr &= ~(1);
        dregs->dma_addr0 = (addr >> 24) & 0xff;
        dregs->dma_addr1 = (addr >> 16) & 0xff;
        dregs->dma_addr2 = (addr >>  8) & 0xff;
        dregs->dma_addr3 = (addr      ) & 0xff;
        ctrl_data &= ~(CYBER_DMA_WRITE);
.......
```

cyberstormII.c

```
....
static void dma_dump_state(struct NCR_ESP *esp)
{
        ESPLOG(("esp%d: dma -- cond_reg<%02x>\n",
                esp->esp_id, ((struct cyberII_dma_registers *)
                                (esp->dregs))->cond_reg));
        ESPLOG(("intreq:<%04x>, intena:<%04x>\n",
                custom.intreqr, custom.intenar));
}

static void dma_init_read(struct NCR_ESP *esp, __u32 addr, int
        length)
{
        struct cyberII_dma_registers *dregs =
                (struct cyberII_dma_registers *) esp->dregs;

        cache_clear(addr, length);

        addr &= ~(1);
        dregs->dma_addr0 = (addr >> 24) & 0xff;
        dregs->dma_addr1 = (addr >> 16) & 0xff;
        dregs->dma_addr2 = (addr >>  8) & 0xff;
        dregs->dma_addr3 = (addr      ) & 0xff;
}
.........
```

How to Solve Cloning "Problem"

- Clone management through development process?
  - Unlikely in this case, since it's hard to incorporate into open source development
- Automatic clone detection and removal?
  - Not clear that tools are adequate for "real world" cloning problems
  - Software developed and maintained by different parties
  - Architecture of the subsystem would be "broken"

Proposed Clone Solution

- Combination of clone control and removal:
  - Make driver "template" that separates generic code from driver specific one
  - Clearly indicate which parts of driver are to be changed and which not
  - "Alarm" other developers when bug discovered in common code
- This allows independent development, preserves architecture, and simplifies design
- Applicable to all "plug-in" based software

Conclusion

- It's not clear that current clone detection tools "do the right thing"
- Theory developed on clone management, detection, and removal is not universally applicable to all types of applications, languages, and designs
  - Need more qualitative analysis of "cloning in the real world"
- Combination of different approaches should give the best results

Ongoing & Future Work

- More detailed *qualitative* analysis of "cloning in the real world"
- More investigation of relative effectiveness of clone detection tools
- Investigation of "parallel evolution" by maintenance type
  - bug fixes
  - new features
  - restructuring
- Investigate another driver family, see if results are similar *e.g.*, Linux network card drivers