Measuring Software Changes with CLOC

S.A.F.E.

Software Analysis & Forensic Engineering Corp.

About Bob Zeidman

- President of Software Analysis & Forensic Engineering Corp.
- President of Zeidman Consulting
- Author of *Designing with FPGAs and CPLDs*, *Verilog Designer's Library*, *Introduction to Verilog*, and articles on engineering and business
- Degrees from Cornell and Stanford

Agenda

- The evolution of software projects
- Traditional methods of measuring software evolution
  - Lines of code
  - Complexity
- CLOC definition
- Open source results
  - Linux
  - Apache
  - Firefox

Importance

- Measuring progress of long-term projects
- Improved maintenance of code
- Measuring work done under contract
- Performing due diligence before acquiring software or software companies
- Valuation of intellectual property
- Transfer pricing

Traditional Methods

- Source Lines of Code (SLOC)
  - Physical SLOC (LOC)
  - Logical SLOC (LLOC)
- Halstead Measure
  - Volume
  - Mental Effort
- Cyclomatic Complexity (McCabe)
- Function Point Analysis

Source Lines of Code (“SLOC”)

- Count the number of lines of code
- Simple
- More source code = more effort
- General measure

Physical SLOC ("LOC")

- Count all lines
- Count blank lines?
- Programming language independent
- Very simple to implement
- Influenced by formatting
- Counts comments equally with functional statements
- Cannot measure changes
- Refactoring = deterioration

Logical SLOC (“LLOC”)

- Count all functional statements
- Programming language dependent
- Complex to implement
- Does not count comments at all
- Cannot measure changes
- Refactoring = deterioration

SLOC Issues

```
for(i = 0; i < 100; i += 1) printf("hello");
```

- LOC: 1
- LLOC: 2 ?
  - `for (i=0; i < 100; i += 1)`
  - `printf("hello");`
- LLOC: 4 ?
  - `i=0;`
  - `while i < 100;`
  - `i += 1;`
  - `printf("hello");`

Halstead Measure

- Maurice Halstead, 1977
- n1 = number of unique operators
- n2 = number of unique operands
- N1 = number of operator occurrences
- N2 = number of operand occurrences

Halstead Measure

- Program length
  - $N = N1 + N2$
- Program vocabulary
  - $n = n1 + n2$
- Volume
  - $V = N \cdot \log_2 n$
- Difficulty
  - $D = (n1 \cdot N2)/(2 \cdot n2)$
- Effort
  - $E = D \cdot V$
- Time
  - $E / 18$

Halstead Measure

- Programming language dependent
- Simple to implement
- Modern languages
  - `i = i + 1;`
  - `i += 1;`
  - `a = sqrt(i);`
  - `int i;`
  - `*ptr = &abc;`
  - `public class MyClass`
- Comments not counted
- Usefulness?

Cyclomatic Complexity

- Thomas McCabe, 1977
- Count the number of individual paths through the code
  - *printf* statement has one path
  - *if* statement has two distinct paths
    - one path for *true*
    - one path for *false*
  - *case* statement has multiple distinct paths
- Calculated by creating graphs and counting execution paths
- Complex to implement
- Complexity ≠ source code evolution
- Refactoring = reduction of complexity

Function Point Analysis

- Allan Albrecht, IBM 1979
- Categories
  - Outputs
  - Inquiries
  - Inputs
  - Internal files
  - External interfaces
- Assessed for complexity
- Assigned a number
- Adjustments and weighting

Function Point Analysis

- Very labor intensive
- Very expensive
- Requires certified function point consultants
- Requires a fairly complete set of requirements

Function Point Analysis

- IFPUG function points
- Backfired function points
- COSMIC function points
- Finnish function points
- Engineering function points
- Feature points
- Netherlands function points
- Unadjusted function points
- Function points light

Source Code Differentiation

- The measure of the number of LOC that match completely as a fraction of the total LOC
- Value between 0 and 1
- Order of statements is not considered
  - Lines of software source code can be reordered
  - Entire sections can be cut and pasted
  - No change in functionality of program

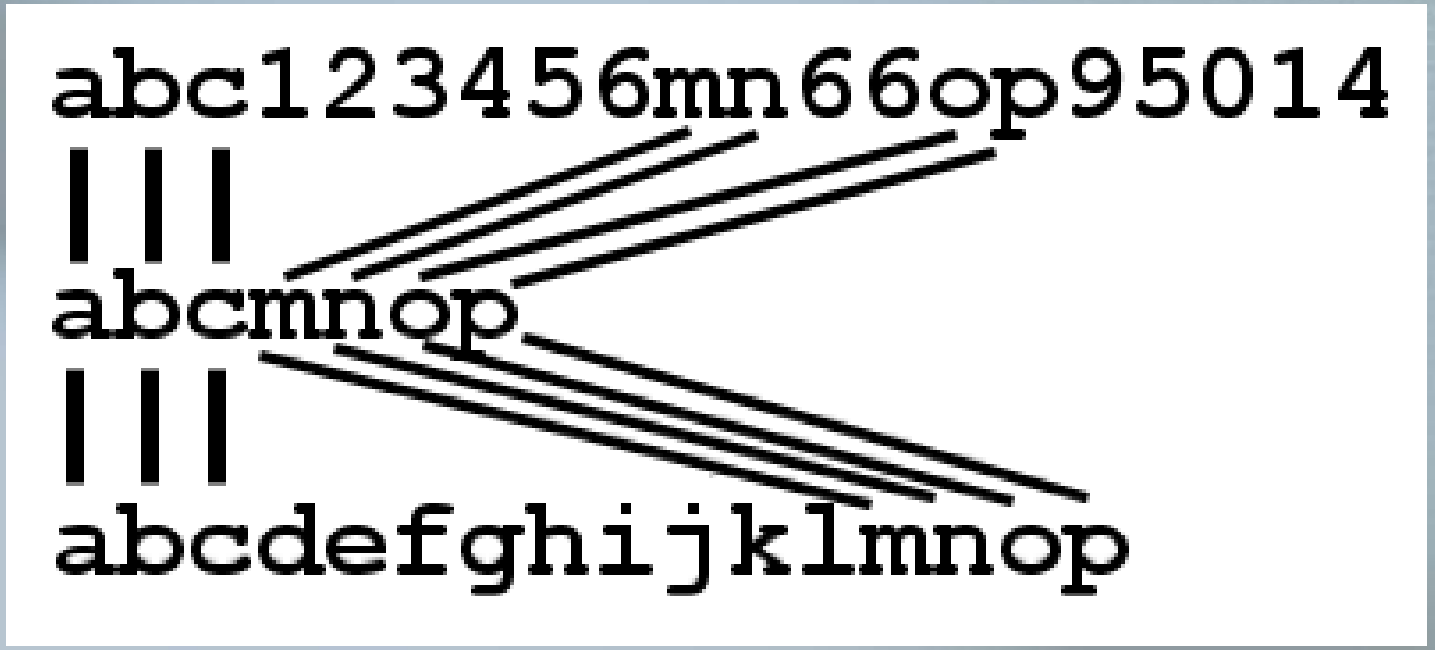
Calculate Similarity Score

- Count the number of lines in files A and B, *L(A)* and *L(B)*
- Determine the number of matching lines, *m(A,B)*
- Matching lines is one-to-one

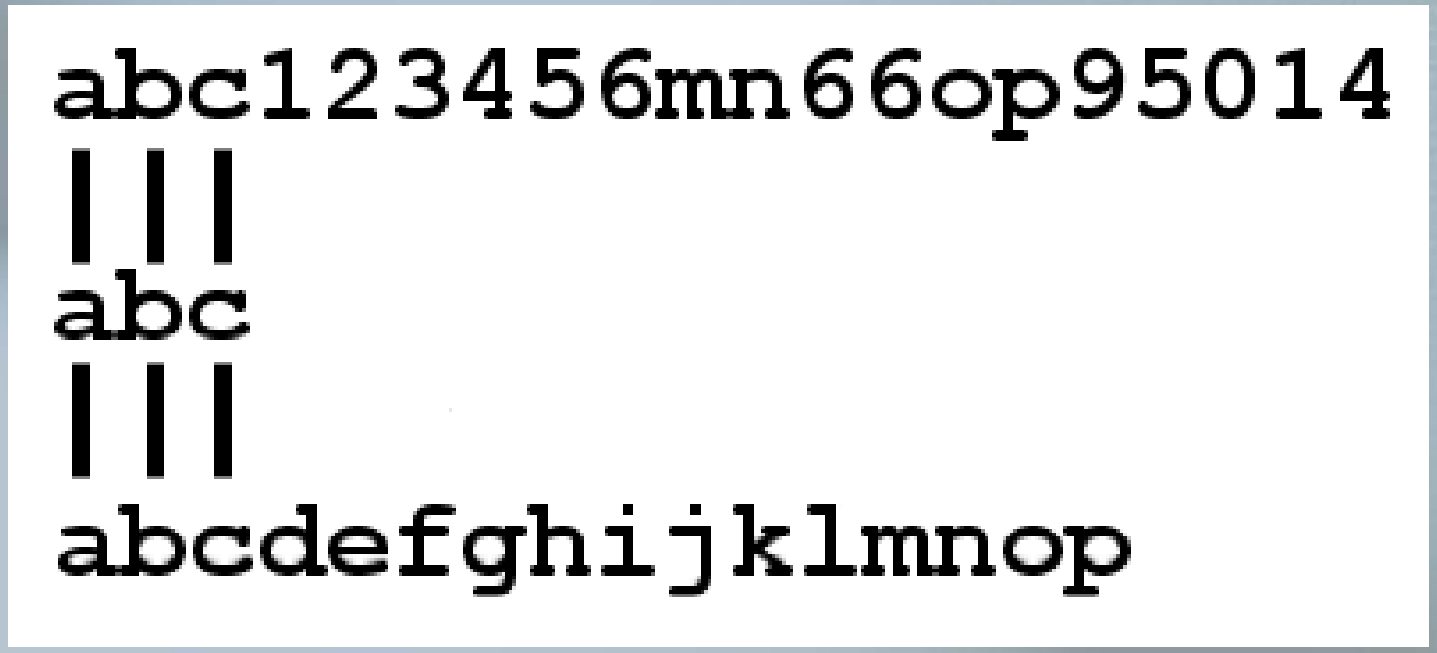
Source Code Differentiation

Symbol	Definition
σ	source code similarity between two files
F(n)	file number *n*
D(n)	location of file number *n* (e.g., directory path)
L(n)	the number of lines in file *n*
M	the number of lines that match between two files
m(i,j)	match score for lines *i* and *j*
l(i)	the length of (i.e., number of characters in) line *i*
w(c)	a character weighting function for character *c*
w(i)	the weighted length of line *i*
$c_k(i)$	character *k* of line *i*
LCS(i, j)	longest common subsequence of lines *i* and *j*
LCCS(i, j)	longest common contiguous subsequence (substring) of lines *i* and *j*
WR(i)	whitespace reduction of line *i*

Longest Common Subsequence



Longest Common Substring



$\sigma(F^n, F^m) = \sigma(F^n, F^m) \; \forall \, D^n, D^m$

Axioms

- Commutativity Property
  - Not dependent on the order of comparison

    σ(F(n), F(m)) = σ(F(m), F(n))

- Identity Property
  - A file cannot be more similar to another file than it is to itself

    σ(F(n)) = σ(F(n), F(n)) = 1

- Location Property
  - Independent of the location of the files

    σ(F(n), F(m)) = σ(F(n), F(m)) for all D(n), D(m)

Mutual Similarity

- Two times the total number of matching lines in file n and file m, divided by the total number of lines in file n and file m

$$\sigma(F(n), F(m)) = 2M(F(n),F(m)) / (L(n)+L(m))$$

Directional Similarity

- Total number of matching lines in two files divided by the total number of lines in one of the files

$$\sigma_n(F(n), F(m)) = M(F(n),F(m)) / L(n)$$

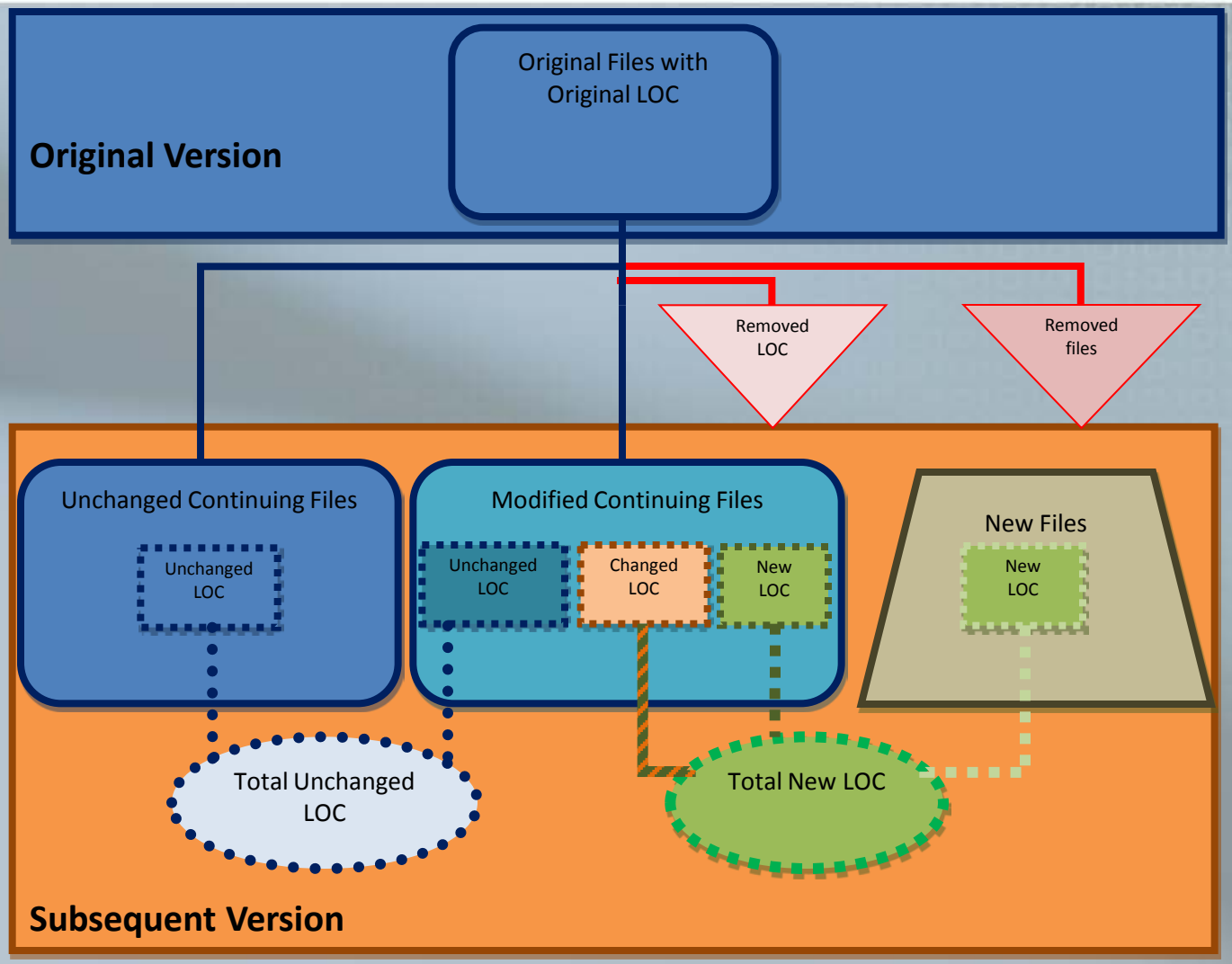
$$\sigma_m(F(n), F(m)) = M(F(n),F(m)) / L(m)$$

CLOC Definition

Changed Lines of Code (“CLOC”)

- Counts the number of lines of code that have been
  - Modified
  - Added
  - Remain constant

CLOC


Original Files with
Original LOC
Original Version
Removed
LOC
Removed
files
Unchanged Continuing Files
Unchanged
LOC
Modified Continuing Files
Unchanged
LOC
Changed
LOC
New
LOC
New Files
New
LOC
Total Unchanged
LOC
Total New LOC
Subsequent Version

CLOC Methodology

- CodeMeasure®
- Software Analysis & Forensic Engineering Corporation ("S.A.F.E.")
  - Compare versions
  - CLOC Spreadsheet

CLOC Measures

- **Growth**: percentage of total new LOC to the total LOC in the original version
- **LOC Decay**: percentage of total continuing LOC to the total LOC in each subsequent version.
- **File Decay**: percentage of original files that are still remaining to the total number of files in each subsequent version

CLOC Definitions

- *TF(n)*: Total Files
  - Total number of files in version *n* of the program
- *NF(n,m)*: New Files
  - Number of new files in version *n* from previous version *m*
- *TL(n)*: Total LOC
  - Total lines of code in version *n*
- *TNL(n,m)*: Total New LOC
  - Total new lines of code in version *n* from previous version *m*
- *TCF(n,m)*: Total Continuing Files
  - Total files in version *n* that also existed in previous version *m*
- *TLCF(n,m)*: Total LOC in continuing files

CLOC Definitions

- ***NLCF(n,m)***: New LOC in Continuing Files
  - LOC in continuing files but that are in the file in new version *n* but not in the file in previous version *m*
- ***CL(n,m)***: Continuing LOC
  - LOC present in the files in previous version *m* and in the continuing files in version *n*
- ***MCF(n,m)***: Modified Continuing Files
  - Number of files in version *n* that also existed in previous version *m* but that have changed
- ***UCF(n,m)***: Unchanged Continuing Files
  - Number of files in version *n* that also existed in previous version *m* without changes

CLOC Definitions

- ***CLOC Growth(n,m)***
  - Ratio of new LOC in version *n* to LOC in previous version *m*
- ***File Continuity(n,m)***
  - Ratio of continuing files in version *n* from version *m* to total files in version *n*
- ***File Decay(n,m)***
  - Ratio of files in version *n* that are not continuing from version *m* to total files in version *n* (1- File Continuity)

CLOC Definitions

- ***Line Continuity(n,m)***
  - Ratio of continuing LOC in version *n* from version *m* to total LOC in version *n*
- ***Line Decay(n,m)***
  - Ratio of LOC in version *n* that are not continuing from version *m* to total LOC in version *n* (1 – Line Continuity)
- ***Unchanged File Continuity(n)***
  - Ratio of unchanged continuing files in version *n* to total files in version *n*

CLOC Equations

- CLOC Growth(n) = TNL(n) / TL(0)
- File Continuity(n) = TCF(n) / TF(n)
- File Decay(n) = 1 – (TCF(n) / TF(n))
- Line Continuity(n) = CL(n) / TL(n)
- Line Decay(n) = 1 – (CL(n) / TL(n))
- Unchanged File Continuity(n) = UCF(n)/ TF(n)

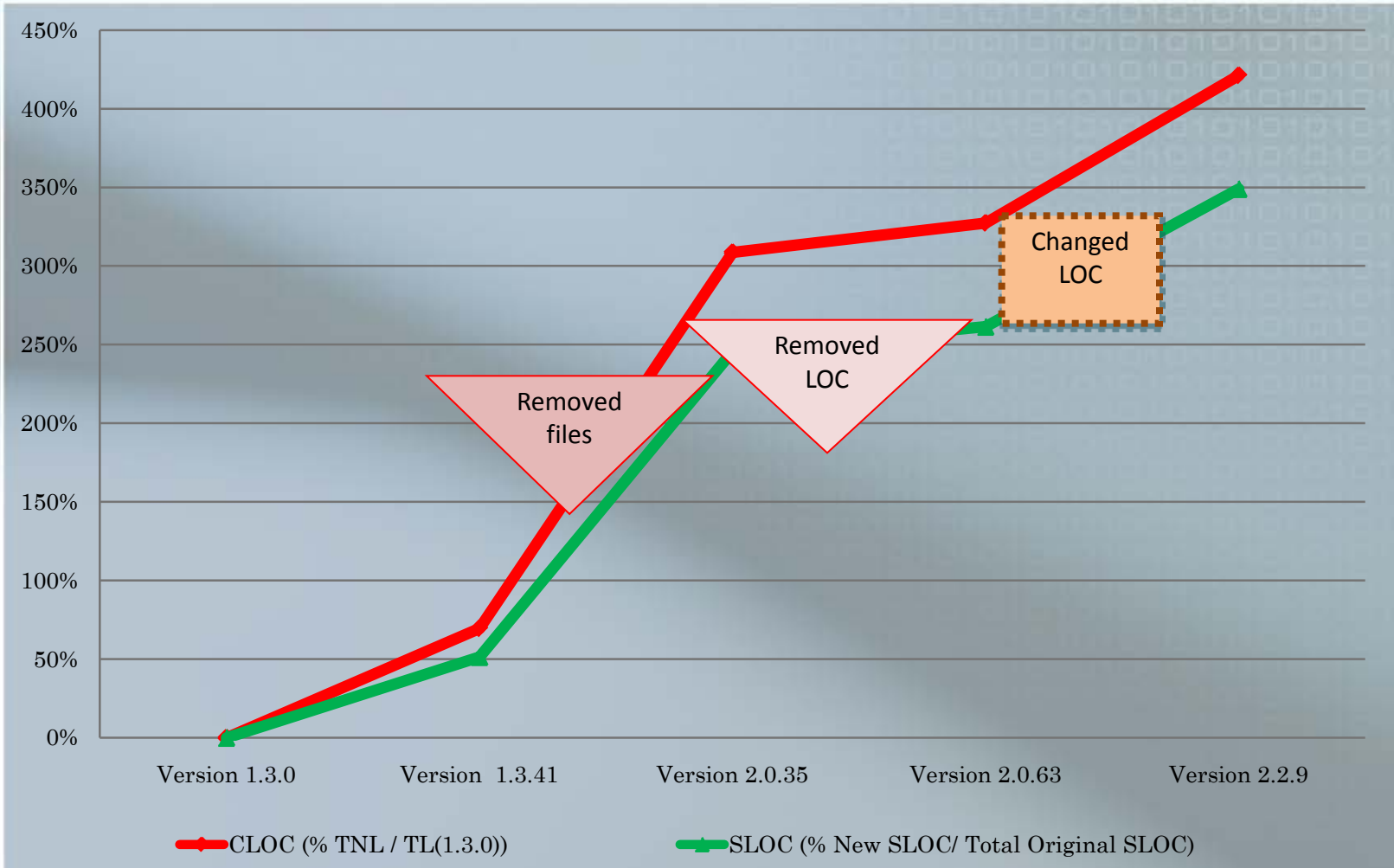
Three Open-Source Projects

- Linux Kernel
- Apache HTTP Server
- Mozilla Firefox

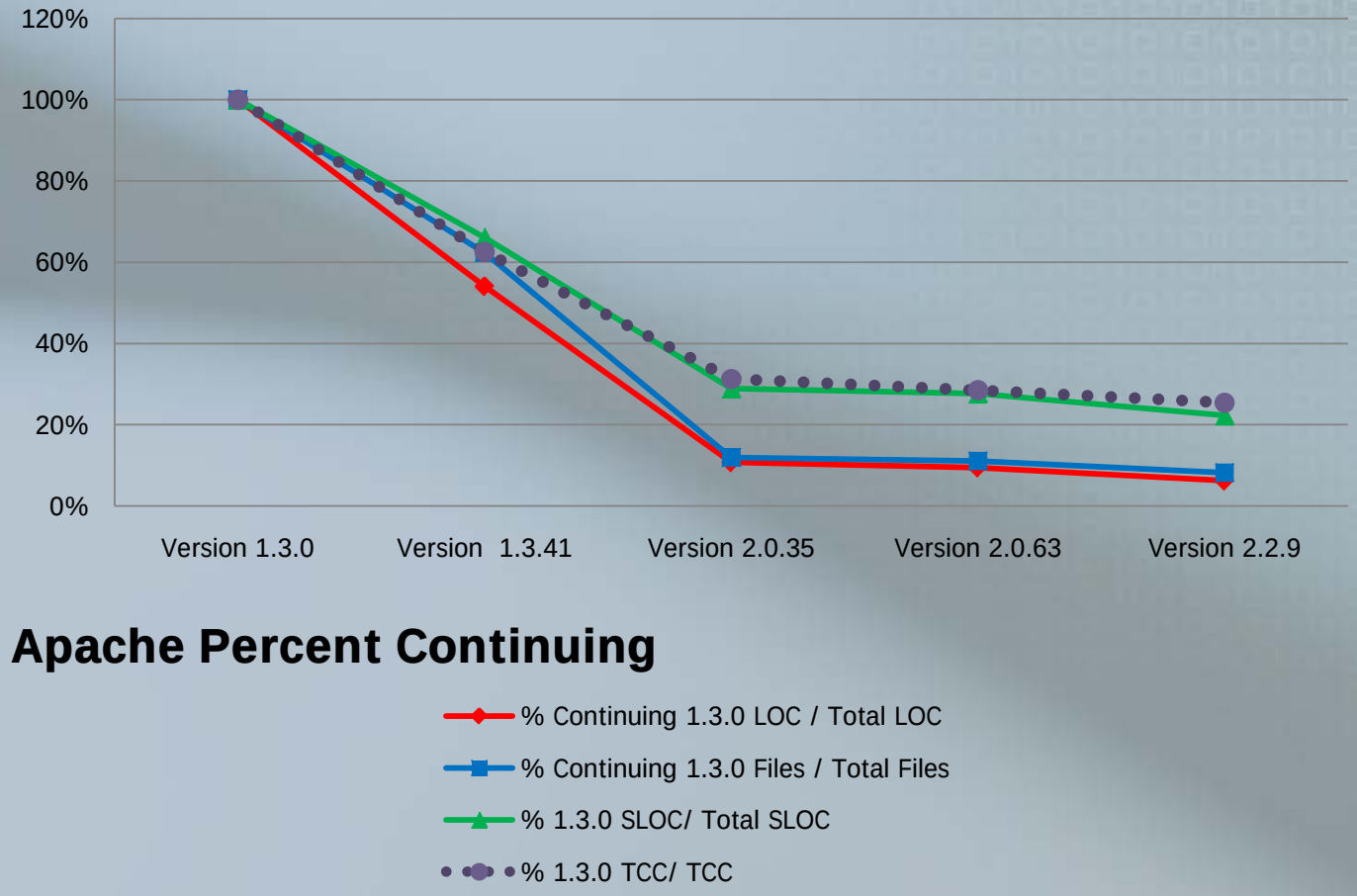
Mozilla Firefox Browser

Data from Firefox browser	Version .1	Version 1.0	Version 2.0	Version 3.0
Total Files: TF(n)	10,302	10,320	11,042	9,429
New Files: NF(n)	0	175	1,455	2,399
Total LOC: TL(n)	3,169,530	3,180,268	3,570,712	3,288,983
Total New LOC: TNL(n)	0	40,922	843,683	1,530,918
Total Continuing Files: TCF(n)	10,302	10,145	9,587	7,030
Total LOC in Continuing Files: TLCF(n)	3,169,530	3,148,460	3,125,785	2,288,903
New LOC in Continuing Files: NLCF(n)	0	9,114	398,756	530,838
Continuing LOC: CL(n)	3,169,530	3,139,346	2,727,029	1,758,065
Modified Continuing Files: MCF(n)	0	281	8,577	6,543
Unchanged Continuing Files: UCF(n)	10,302	9,864	1,010	487
SLOC Growth(n)	**0%**	**0%**	**13%**	**4%**
CLOC Growth(n)	**0%**	**1%**	**27%**	**48%**
File Continuity(n)	**100%**	**98%**	**87%**	**75%**
Unchanged File Continuity(n)	**100%**	**96%**	**9%**	**5%**
LOC Continuity(n)	**100%**	**99%**	**76%**	**53%**

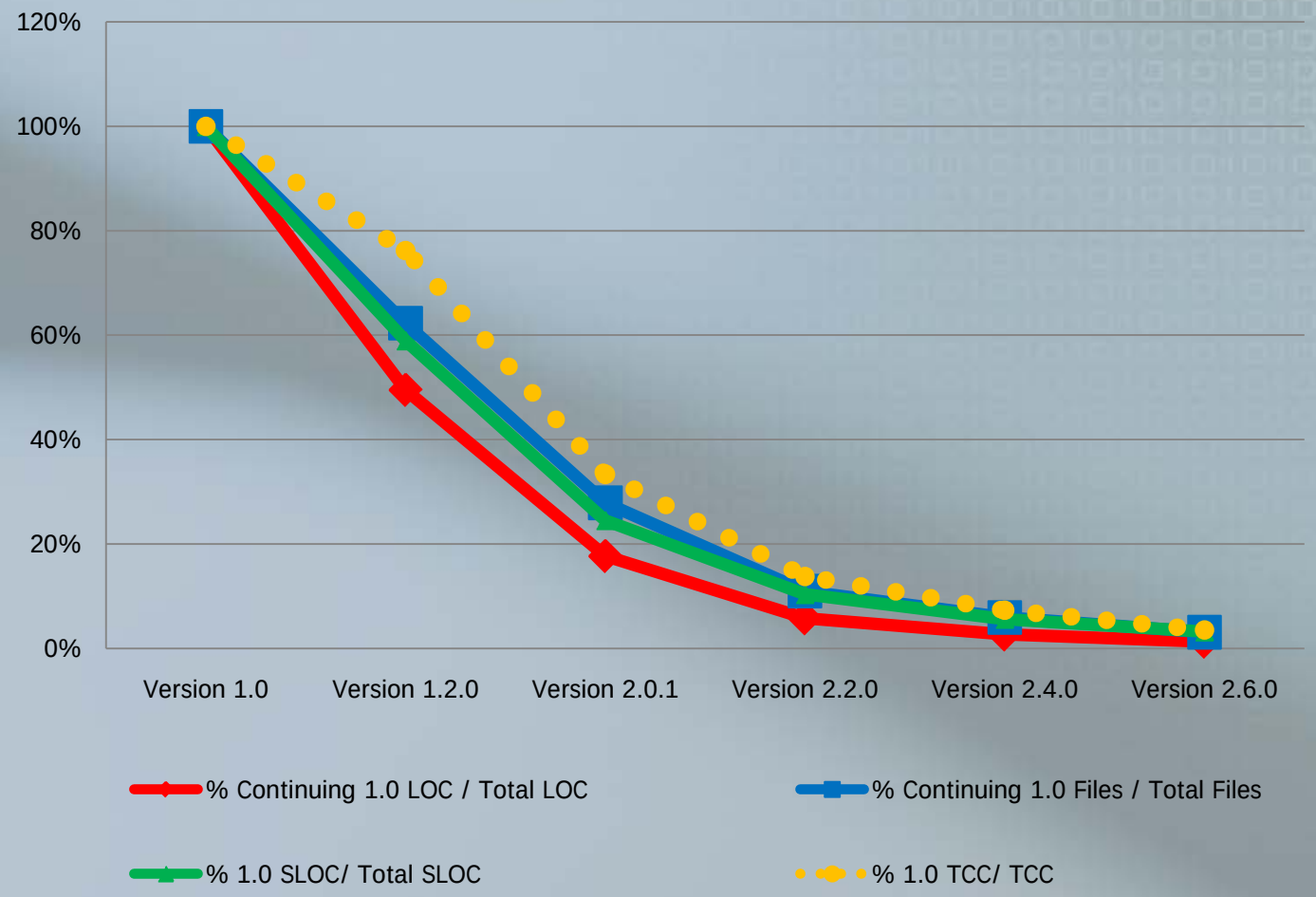
Apache Growth



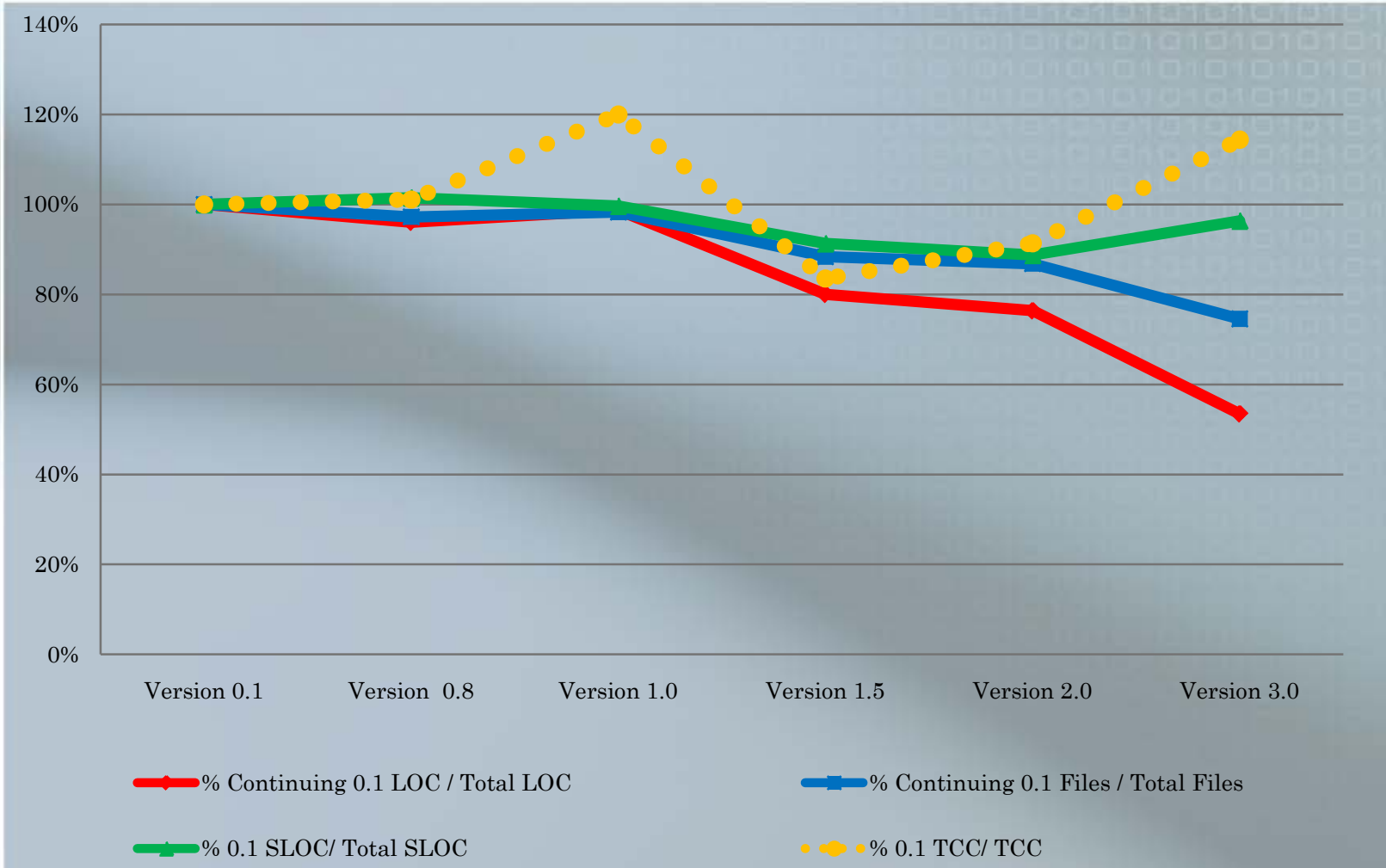
Apache LOC and File Decay



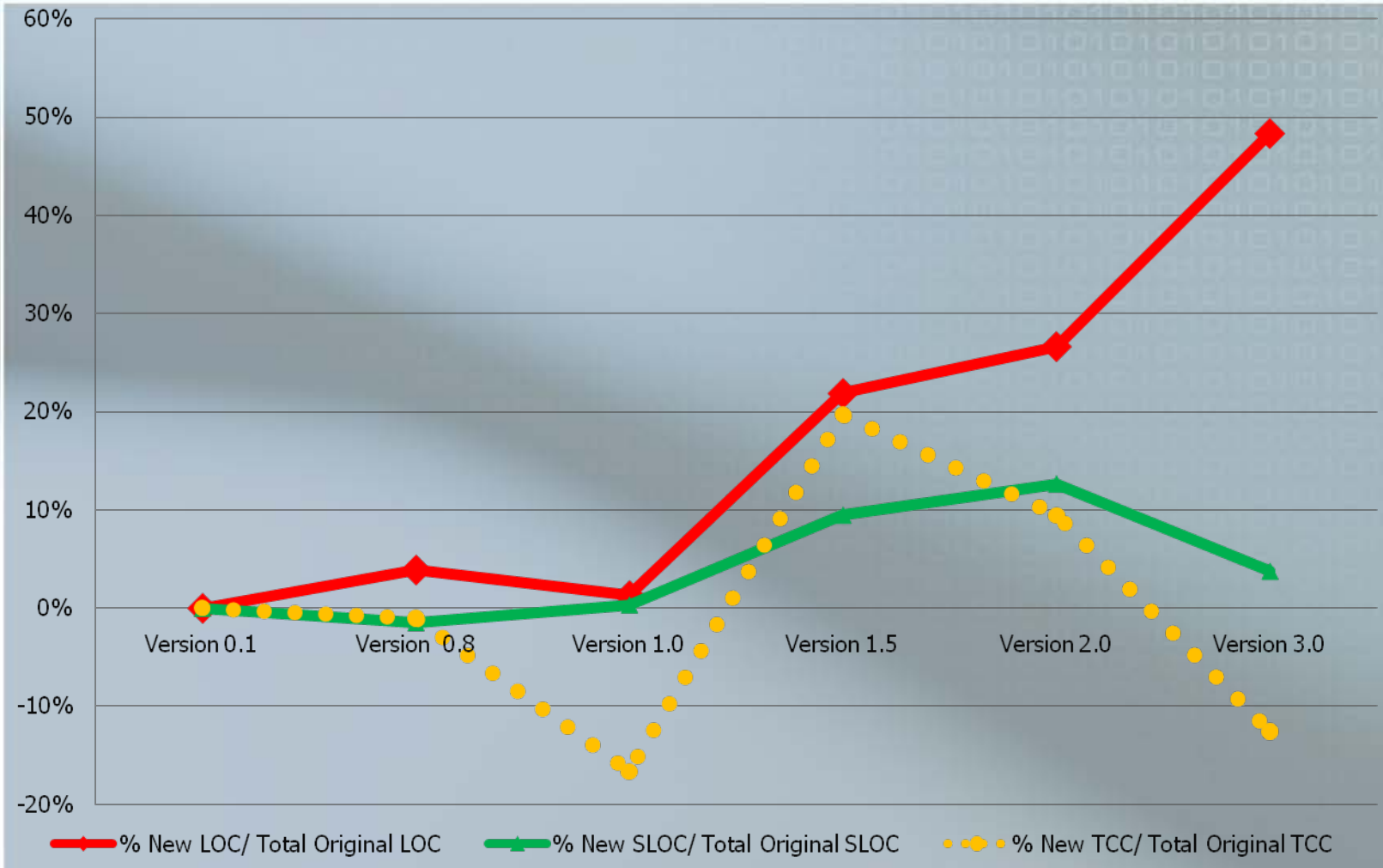
Linux LOC and File Decay



Firefox LOC and File Decay



Firefox Growth



Summary

- Traditional SLOC is static and imprecise
- Halstead - questionable
- Function points - complex, expensive, not connected to code evolution
- Cyclomatic Complexity - complex, not connected to code evolution
- CLOC shows evolution
  - Quantitative
  - Easy to implement
  - Programming language independent
  - Accounts for added, deleted, and changed lines

Contact Information

Bob Zeidman

Software Analysis & Forensic Engineering

Web: www.SAFE-corp.biz
www.CodeMeasure.com

Phone: +1-408-517-1194

Fax: +1-408-741-5231

Email: bob@SAFE-corp.biz