Gathering Software Metrics from Software Version Control Systems and Automated Build Systems

Mark Cheung
Computer Systems Lab 2009-2010

Abstract

Software metrics are tools that can be used to measure in quantity the software development. They are often used to estimate the cost and resource requirements, the productivity, the data collection, the code quality, and other software performances. The purpose of this project is to develop tools that support the automated production of software metrics from version control systems such as Subversion and build systems like CruiseControl. The project produces code count programs that generate the physical and logical source lines of codes (SLOC) for Java and Midas code. The project will?? also includes the development of tools that assist in generating reports with the COCOMO model and performing analysis of the results.

Introduction

In 2001, Red Hat Linux 7.1 was found to hold 30,152,114 Physical Source Lines of Code. Using the Basic COCOMO model, this figure implies 7955.75 person-years development effort, making a total of 6.53 years and $1,074,713,481 cost (Wheeler). Although these figures are estimates and assumptions were made in calculation, the resulting Physical SLOC can still give us insights into the Linux 7.1 system.

Estimating software cost requires careful calculation and analysis of the software project. A mistake in evaluation can destabilize the development by allocating either too much or too little resources.

Background

Source Lines of Code (SLOC) is a code metrics for counting the number of lines for a set of programs and thereby estimating the amount of effort required. There are various types of counting that can be done, including total lines, commands, compiler directives, executive statements, non-blank lines, physical lines, logical lines, tokens. Tokens are operators or operands such as “while”, and “eof.” SLOC measures are divided into two major types: physical and logical.

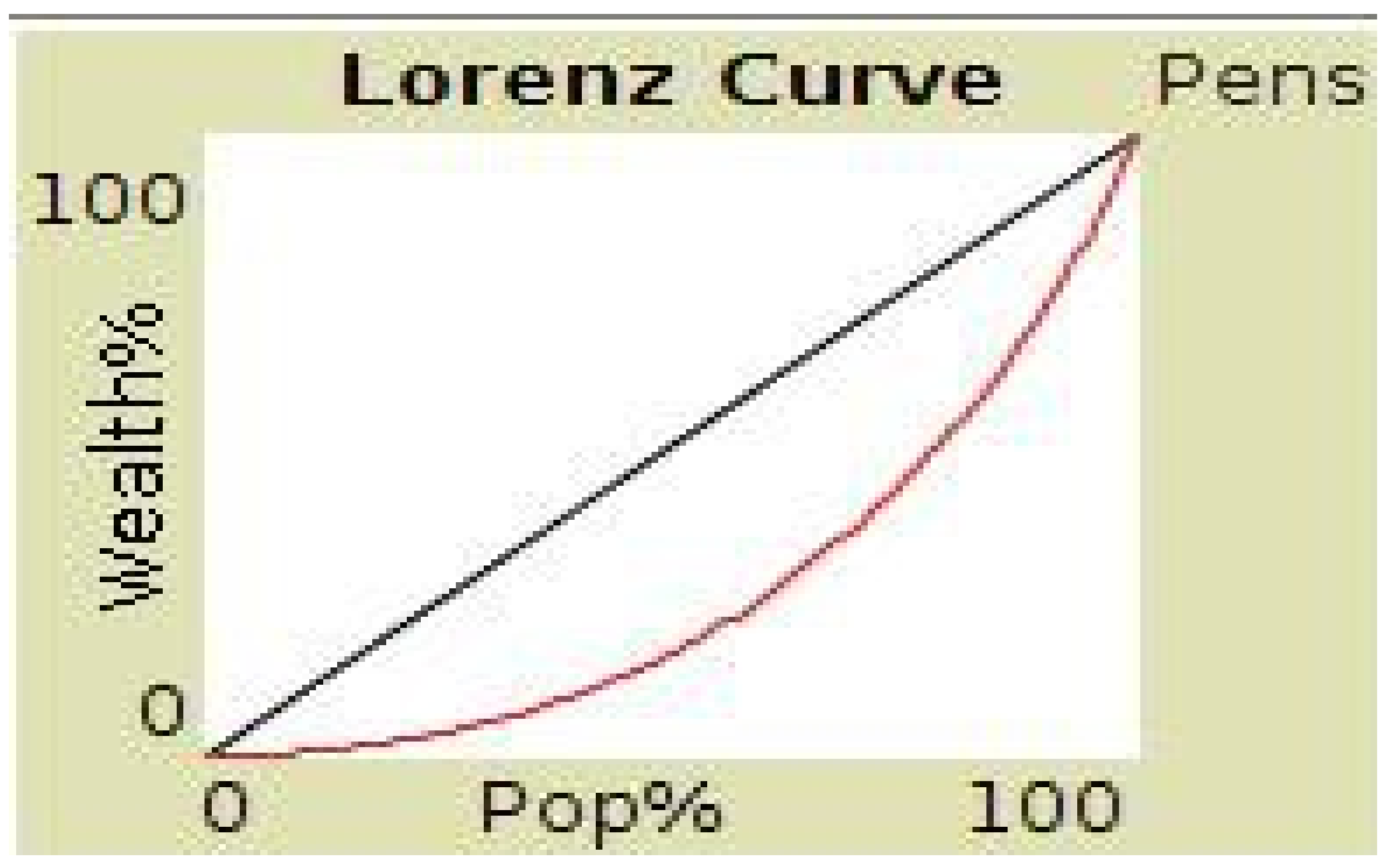

Fig 1: label the figures

Discussion of Methods

1) Evaluate the Northrop Grumman Ultimate Code Line Accumulator Tool (based on USC CodeCount)
2) Extend CodeCount language support as necessary with C: (i.e. X-MIDAS and NextMIDAS scripting languages, Java, Python) (done)
3) Enhance CodeCount as necessary to support counting of auto-generated source code (done)
4) Select/Develop tool to perform automatic production of metrics and store them in a database (Hackystat)
5) Develop single-source plugins for Eclipse RAP and Eclipse RCP with Java to support report generation and analysis of metrics database.
6) Extend metrics collection to include code-quality, code-reuse, code-churn data, build failures, etc.

Results and Conclusions

The focus of your project. What you conclude, what it means.

Mentored by Mr. Michael Ihde, Northrop Grumman

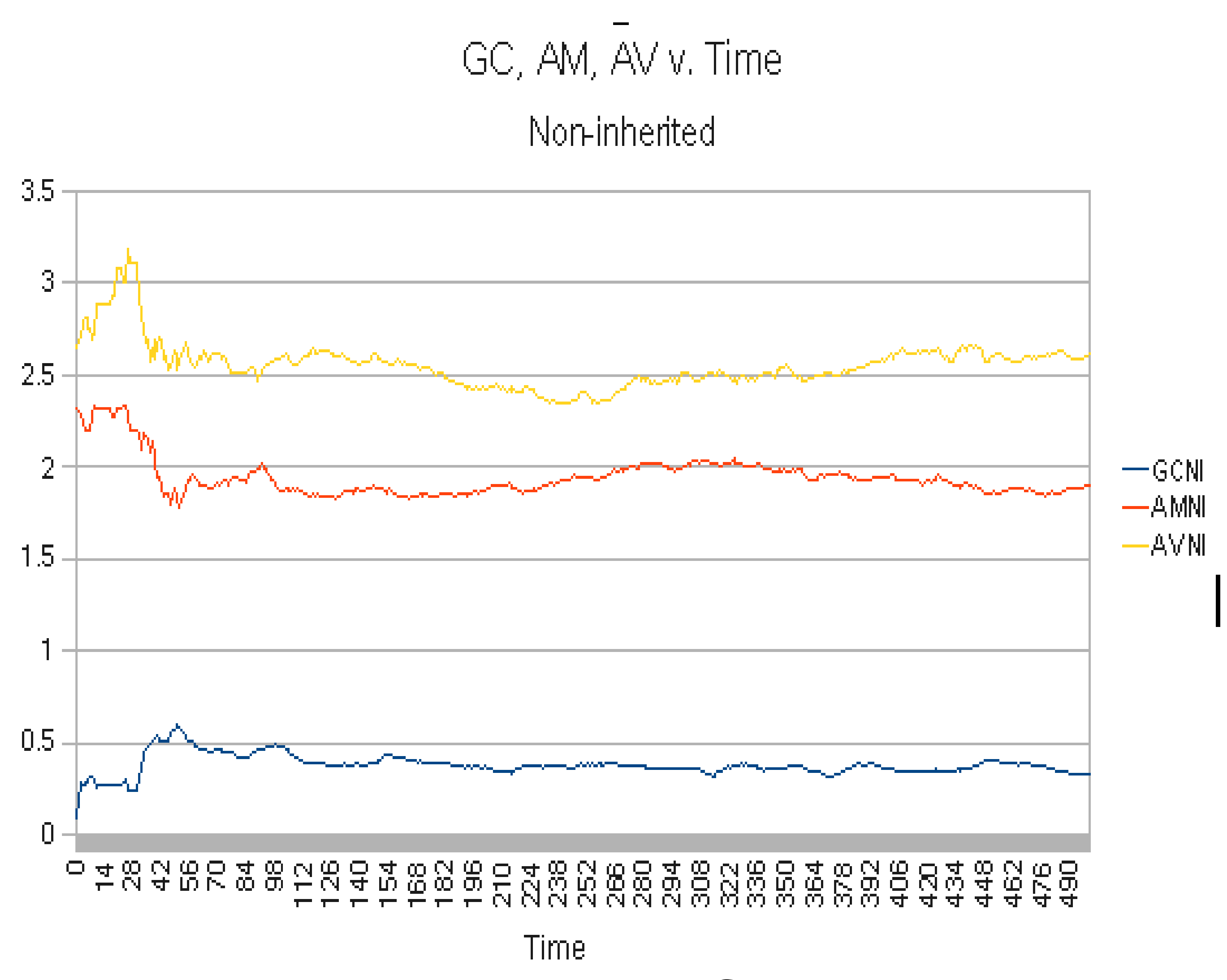

Fig 2: Label the figures. Make sure the figures can be read.

	SCLC	CodeCount total	CodeCount generated	CodeCount (not generated)			
		Physical SLOC	Logical SLOC	Physical SLOC	Logical SLOC	Physical SLOC	Logical SLOC
number of files	650	650	650	x	x	x	x
total lines	170150	170150	146808	110928	92930	59222	53878
blank lines	16252	16252	16252	9525	9525	6727	6727
comments	86665	86210	86210	59469	59469	26741	26741
embedded	x	441	441	328	328	113	113
compiler directives	x	7321	7321	0	0	7321	7321
data declarations	x	12256	4512	7980	2687	4276	1825
executive instructions	x	48111	32513	33954	21249	14157	11264
non-comment source lines	67669	67688	44346	41934	23936	25754	20410
Auto-Generate-Source-Line	47114	x	x	x	x	x	x
asembly-equivalent-source-lines	406014	x	x	x	x	x	x