The Role of Community Source Software in University Business Models

Charles W. Butler
Colorado State University

Don Albrecht
Colorado State University

The Kauli Foundation provides open source community software for universities. The foundation is comprised of many universities, colleges, and commercial firms who have joined together to develop an application portfolio for higher education administration. This research studied the software quality of the Financial System which is one of eight Kauli applications. Specifically, four Financial System modules were analyzed including Budget Construction (BC), Capital Asset Builder (CAB), Capital Asset Management (CAM), and Purchasing/Account Payable (PUR/AP). The analysis of software quality utilized NASA as a reference model. The NASA reference model uses three software metrics for determining software quality and reliability.

INTRODUCTION

Today's large businesses rely on enterprise resource planning (ERP) to enable business processes and to provide integration and end-to-end processing. Two tier-1 vendors, SAP and Oracle, dominate the ERP market, and SAP has 35 years of ERP implementation experience. While ERP solutions are widely popular for business integration solutions, the functionality of these commercial ERP solutions do not fit the unique business processes of universities. SAP and Oracle software modules enable business processes such as manufacturing, logistics, and generally accepted accounting principles while university business models include functions such as class scheduling, registration, funded research administration, and grant management. The university community has searched for alternate solutions for its unique business requirements.

This paper describes the growing market of open community software offered through the Kuali Foundation. Indiana University is the founding partner in Kuali. Today, a growing community of 67 universities, colleges and commercial firms have joined together to form the Kuali Foundation Community Members. These organizations are building and sustaining open-source software for higher education, by higher education. In addition, this paper presents the results of analyzing Kuali software quality using contemporary software metrics. The results of this analysis are presented to provide an indication of whether the outcome of the consortium development strategy is one that is risky for a university.

BACKGROUND

The complexity and cost of tier-1 commercial ERP implementation are major constraints on universities. An SAP or Oracle implementation can cost a business anywhere from $50M to $1B,

depending on the breadth and depth of ERP deployment. This financial commitment is generally beyond the project investment expenditures of most universities. As a result, universities typically deploy software solutions using various acquisition methods. Universities have at their disposal several methods of software acquisition that may be mixed and matched into hybrid solutions. They are as follows:

- In-house developed
- Commercially developed
- Open source developed
- Community source developed

In-House Developed

In-house developed is software that is designed, created and supported by either the institution's in-house programming staff or contract programmers. It affords the greatest level of institutional control in exchange for having a very high lifecycle cost. This cost lies mainly with fact that the needed application expertise is supported entirely by the individual institution. (Shelly & Harry, 2010)

Commercially Developed

Commercially developed and distributed software is often referred to as commercial off the shelf software (COTS). SAP and Oracle are types of COTS. They require the least amount of institutional technical expertise. It is thought to have the least costly lifecycle cost. This benefit is gained in exchange for the surrender of institutional control over its design and upgrade path. In addition, the institution risks losing future vendor support by not following the vendor's upgrade path or the vendor choosing to discontinue supporting their application. (Shelly & Harry, 2010)

Open Source Developed

Open source development is best thought of as "peer production" by multiple collaborators. The end-product is an application and appropriate design documentation that are available at no cost. It gives the participating institution a free starting point that allows development control at a low initial cost. Unlike in-house developed software, there is a support community available outside the institution. Unfortunately, it is a community of independent developers with different agendas and differing approaches to code development. Any significant institutional enhancement efforts still run the risk of high maintenance costs, design creep, and a possible lack of expertise when support issues are encountered. (Aberdour, 2007)

Community Source Developed

Community source development differs from open source in that it is sponsored by a community of institutions. This community charges membership fees and enforces best programming practices. (Technical Council, 2007) Most members contribute developers to the effort. This acquisition method is a community of developers with similar agendas and an agreed upon approach to code development. Thus, the community source model provides a more solid support structure, rather than purely volunteer efforts as found in strictly open source communities. High maintenance costs and design creep are still a very real risk. (Wheeler, Open Source 2010 Reflections on 2007, 2007) As the need for an enterprise, integrated software solution continues to rise, university business models are increasingly turning to the use of community source developed software. They see this tactic as a benefit in part because: (Wheeler, Open Source 2010 Reflections on 2007, 2007)

- It provides more control over features provided, development priorities, and an upgrade path than COTS.
- Its initial cost is often less than COTS.
- It is less expensive than strictly in-house developed software.
- It is more scalable than open source development.
- It has more ready access to consistent technical support than open source.

KUALI FOUNDATION

As previously written, the Kuali Foundation is a consortium of interested universities, colleges and commercial firms have joined together to produce an enterprise software solution for the academic business model. The Kauli Foundation utilizes a community source developed software acquisition method. TABLE 1 contains a number of the Carnegie Mellon class universities participating in the foundation. In addition, other leading commercial firms such as IBM and VMware are participants.

TABLE 1
CARNEGIE MELLON CLASS KAULI UNIVERSITY MEMBERS

Boston College	Michigan State University
Boston University	University of Arizona
Clemson University	University of Arkansas
Colorado State University	University of California - Berkeley
Cornell University	University of Florida
Clemson University	University of Hawaii
Colorado State University	University of Illinois
Cornell University	University of Michigan

The consortium pools resources to develop and sustain many of the software systems needed for higher education. This approach reduces costs and produces software that better fits institutional needs. The Kuali Foundation is funded through a fee-based membership starting at $4,500 growing to $24,500 based upon university budget size. These universities share information technology (IT) resources to develop software and the software can be used by anyone without a purchase or maintenance fee. There are 15 lead universities who share IT resources and coordinate project activity as numerous "virtual" projects. These institutions freely share work with the world as that fits the public service mission of colleges and universities. Working collaboratively together, the outcome is shared best practices and reduced costs beyond legacy approaches to purchased software.

Kuali Software

The Kuali software is a portfolio of applications ranging from financial to mobile connectivity. Below is a short summary of each module:

- Financial System (2005): Financial software that meets the needs of Carnegie Class institutions
- Coeus (2006): Research administration for grants administration to federal funding agencies
- Student (2007): Business needs of students, faculty and institutions throughout the academic lifecycle
- Rice (2007): Middleware products that integrate products allowing applications to be built in an agile fashion
- Open Library Environment (2010): Integration of academic and research libraries for managing and delivering intellectual information
- Mobility (2011): Connect mobile devices to a variety of campus systems
- People for the Enterprise (scheduled 2013): HR/Payroll System built by higher education for higher education
- Ready (scheduled 2013): A business continuity planning tool

These modules are in various stages of development of implementation. Currently, all of the consortium universities and colleges are using or are in deployment phases of the Financial System.

FINANCIAL SYSTEM SOFTWARE ANALYSIS

This research project studied the software quality of the Financial System. The Financial System was chosen due to its wide spread implementation including Colorado State University (CSU). The Financial System consists of 8 modules identified as follows:

1. Account Receivable (AR)
2. Budget Construction (BC)
3. Capital Asset Builder (CAB)
4. Capital Asset Management (CAM)
5. Contract and Grants (CG)
6. Effort Certification (EC)
7. Labor Distribution (LD)
8. Purchasing/Account Payable (PUR/AP)

For the purposes of this study, four of these modules were analyzed - Budget Construction (BC), Capital Asset Builder (CAB), Capital Asset Management (CAM), and Purchasing/Account Payable (PUR/AP). CSU has not yet deployed the other modules.

Code Complexity

Analyzing software quality can be subjective or objective methodology. As a basis for software quality, NASA was chosen as a reference model given the agency's mission critical, high performance standards. The following three metrics are used in NASA's Metrics Data Program (MDP) repository for determining software quality and reliability. (NASA, p. 4)

- Halstead Metrics: Halstead's metrics assumes that a program should be viewed as an expression of language. Halstead expressed mathematically the relationships among the number of variables, the type of programming statements and the complexity of the code. Unfortunately, Halstead metrics are difficult to compute. In order to be useful, a metric must be computed quickly and easily understood. (Kaur, Minhas, Mehan, & Kakkar, 2009)
- McCabe Cyclomatic Complexity: McCabe's cyclomatic complexity measures the number of linearly-independent paths through a program module. The basic assumption is that software complexity is directly related to the number of control paths generated by the code. This metric is easy to understand. In fact, this metric has been criticized because it seems too simple. However, the McCabe metric is an easy-to-compute, high-level measure of a program's complexity which has been shown to agree with empirical data. (Kaur, Minhas, Mehan, & Kakkar, 2009)
- Lines of code metrics: The lines of code metric seems easy to measure. While a longer program can be more prone to error than a short program, that is not always true. In addition, newer object oriented coding techniques make it difficult to determine the actual lines of code executed.

Looking at the competing metrics, McCabe's cyclomatic complexity is an appropriate metric to use to predict software reliability. It has been validated by two NASA studies. (NASA IV&V Facility, 2008). It might also aid in projecting the cost of software support. There are several applications available to measure cyclomatic complexity. However, McCabe IQ is an automated tool that has an array of graphical presentations and is widely accepted by software developers. Therefore, McCabe IQ was chosen as the automated tool for this study. Based upon prior research results, the following McCabe metrics were determined for Kuali Financial Systems modules:

- Cyclomatic complexity: a measurement of the size of a software module's decision logic. Cyclomatic complexity, v, is determined for each software program's modules. Research has shown that when the cyclomatic complexity of a module exceeds 10, then its reliability degrades exponentially. So, a $v > 10$ is considered low quality and riskier software.

- Essential complexity: a measurement of a software module's decision structure or architecture. Essential complexity, ev, is determined for each software program's modules. Research shows that when essential complexity of a module grows higher than 1, it is harder to maintain. So, an ev > 3 is considered to be harder and more difficult to maintain.
- Module design complexity: a measurement of a software module's integration decision structure with other modules. Module design complexity, iv, is determined for each software program's modules. Research shows that when module design complexity of a module grows high, the level of integration testing is increased. More integration results in higher operational risk. So, when iv is a high proportion of a module's decision structure, it is an integration risk.

FINDINGS

Capital Asset Builder (CAB)

The kfs-cab sub-system contains 59 source code files. Parsing by McCabe IQ resulted in 868 modules. Twelve modules had a v > 10. There were no modules with a v > 20. Key profile indicators are:

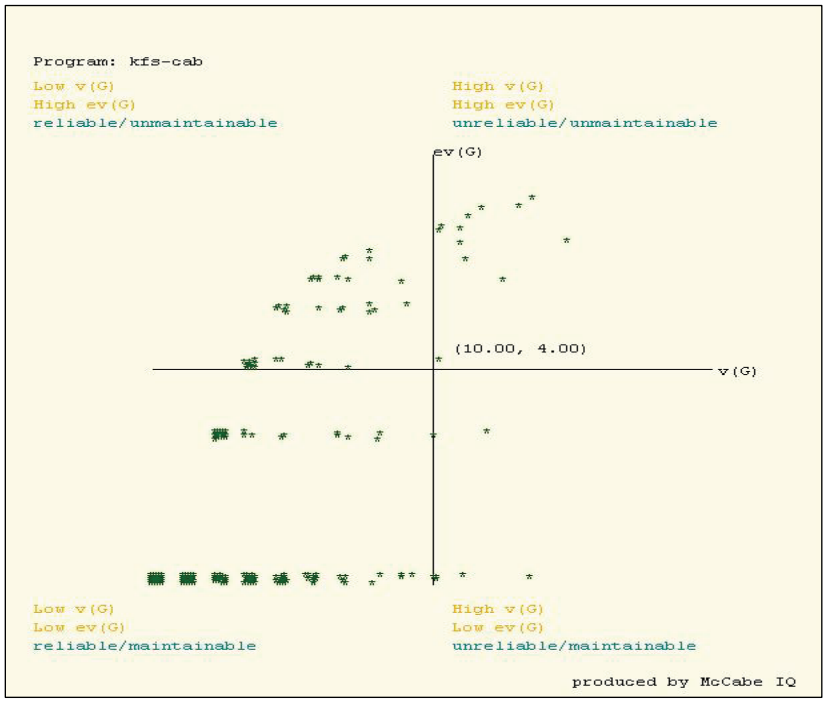
- The average cyclomatic complexity is 2.18.
- The average essential complexity is 1.40.
- The average module design complexity is 2.05.
- The total executable lines of code are 7,477.
- The cyclomatic complexity density is .2525.

A further review of the profile indicators for high risk modules (12 modules with v > 10) yielded the following results:

- The average cyclomatic complexity is 12.8.
- The average essential complexity is 7.6.
- The average module design complexity is 11.2.
- The total executable lines of code are 551.
- The cyclomatic complexity density is .2778.

The cyclomatic complexity results indicate that the kfs-cab sub-system was structured with quality. FIGURE 1 is a scatter plot of cyclomatic and essential complexities. By McCabe metric thresholds, quadrant one contains modules that are unreliable and hard to maintain (1.3% of the modules with v > 10). It would appear that this application is generally reliable & maintainable by the small number of modules in quadrant one. Few errors will be likely when using this module.

FIGURE 1
SCATTER PLOT OF KFS-CAB

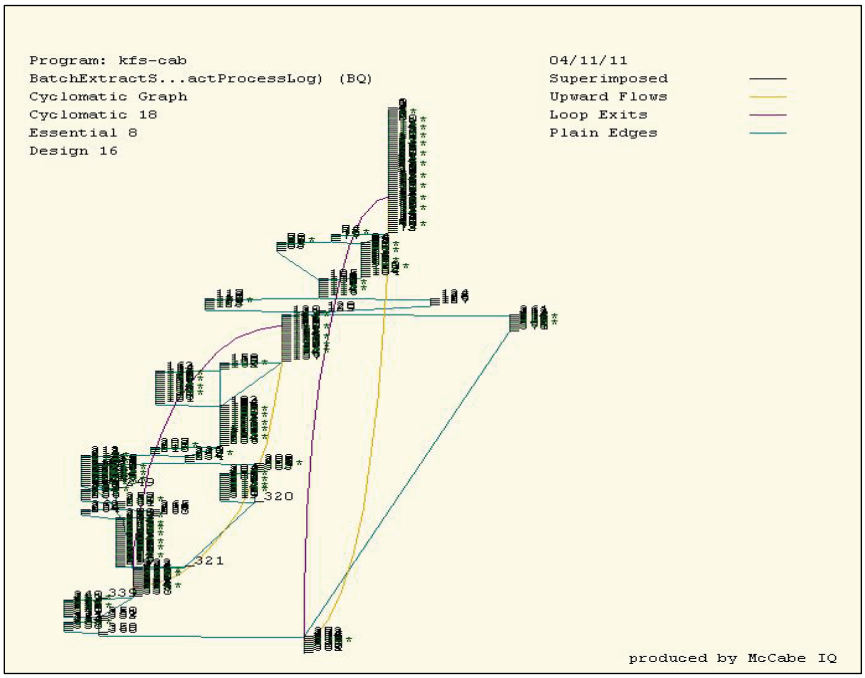


A flowgraph visually represents the logic in a module. It comprises nodes connected by edges. A node designates one of the following program design elements:

- Module entry
- Data element usage
- Call to another module
- Inflow from more than one branch
- Module exit

Nodes and edges are critical to the cyclomatic complexity calculation. A flowgraph that is visually complex merely reflects that the module logic it represents is overly complex. A flowgraph of the most complex module in kfs-cab, as seen in FIGURE 2, illustrates this point.

FIGURE 2
FLOWGRAPH OF A KFS-CAP COMPLEX MODULE



Budget Construction (BC)

The kfs-bc sub-system contains 343 source code files. Parsing by McCabe IQ resulted in 6,266 modules. Forty-five modules had a v > 10. Of these, only five modules had a v > 20. Key profile indicators are as follows:

- The average cyclomatic complexity is 1.41.
- The average essential complexity is 1.11.
- The average module design complexity is 1.37.
- The total executable lines of code are 44,445.
- The cyclomatic complexity density is .1982.

A further review of the profile indicators for modules that had a v > 10 give the following results:

- The average cyclomatic complexity is 15.4.
- The average essential complexity is 6.0.
- The average module design complexity is 14.0.
- The total executable lines of code are 3,764.
- The cyclomatic complexity density is .1849.

The cyclomatic complexity results indicate that the kfs-bc application is high quality. The scatter plot, as shown in FIGURE 3 for kfs-bc, illustrates this quality. It shows, by the small number of modules in quadrant one (45 modules), that most modules are reliable & maintainable.

Application kfs-bc contains five high risk modules with a v > 20. One of these modules, whose cyclomatic complexity is 29, is presented in FIGURE 4. Note the visual complexity of the software logic.

FIGURE 3
SCATTER PLOT OF KFS-BC

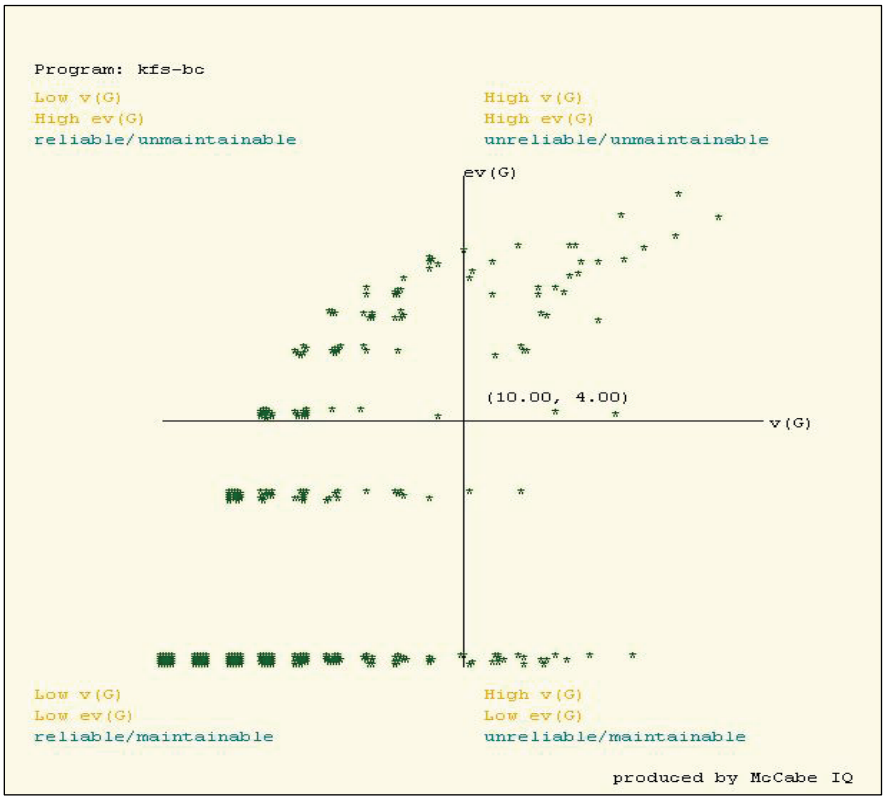
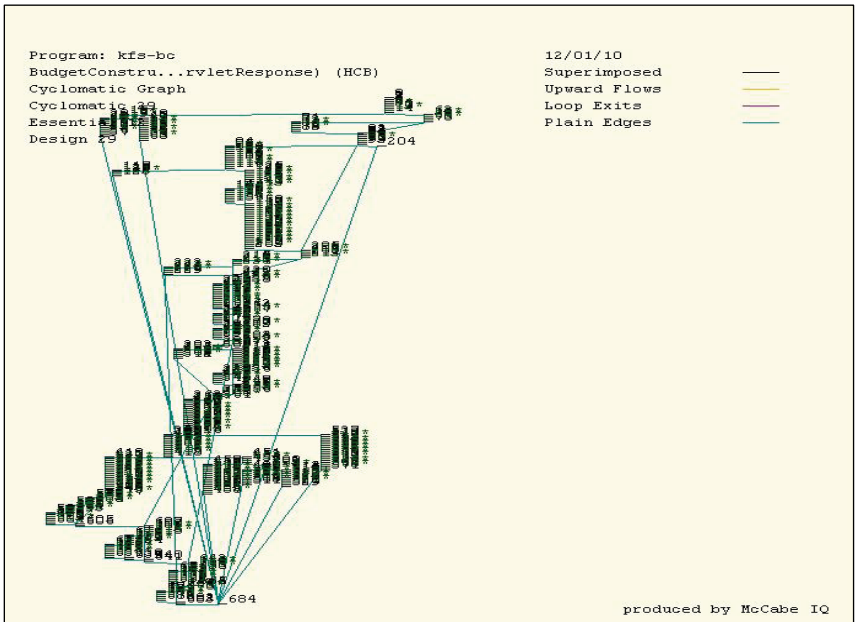


FIGURE 4
FLOWGRAPH OF A KFS-CAB COMPLEX MODULE



Capital Asset Management (CAM)

The kfs-cam sub-system contains 160 source files. Parsing by McCabe IQ resulted in 2,221 modules. Thirty-two modules had a v > 10. Six of the 32 modules had a v > 20. Key profile indicators are as follows:

- The average cyclomatic complexity is 1.83.
- The average essential complexity is 1.26.
- The average module design complexity is 1.74.
- The total executable lines of code are 16,774.
- The cyclomatic complexity density is .2423.

A further review of the profile indicators for modules that had a v > 10 provide the following measurements.

- The average cyclomatic complexity is 16.3.
- The average essential complexity is 5.6.
- The average module design complexity is 14.5.
- The total executable lines of code are 2,450.
- The cyclomatic complexity density is .2135.

The cyclomatic complexity results indicate that the kfs-cam application is high quality. The scatter plot, as shown in FIGURE 5 for kfs-cam, illustrates this quality. It shows, by the small number of modules in quadrant one (32 modules), that most modules are reliable & maintainable.

FIGURE 5
SCATTERPLOT OF KFS-CAM SHOWING A LARGE NUMBER OF RELIABLE MODULES

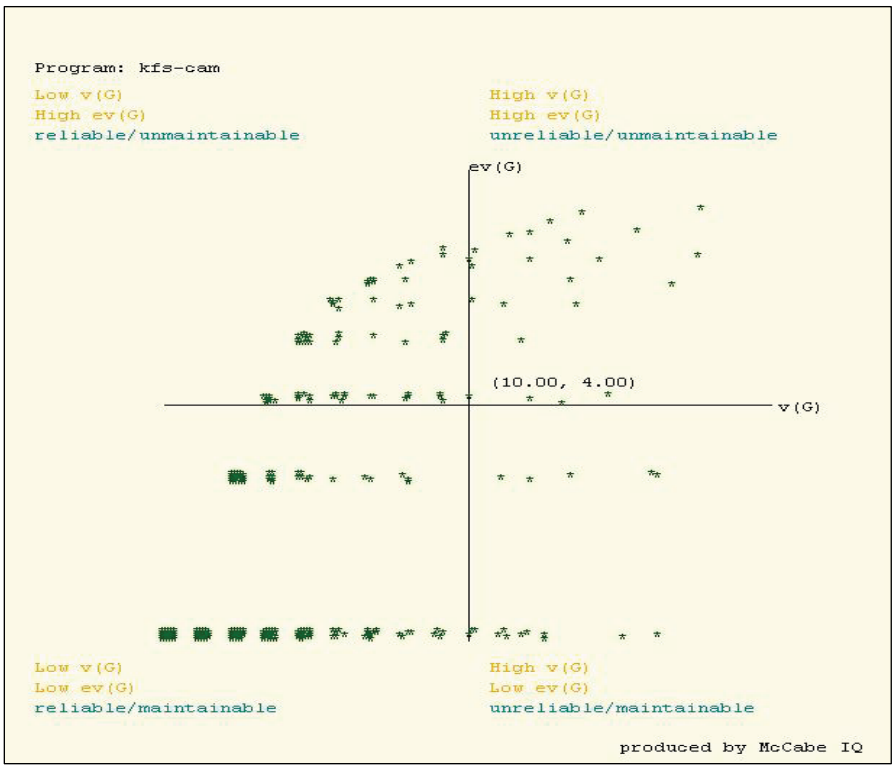
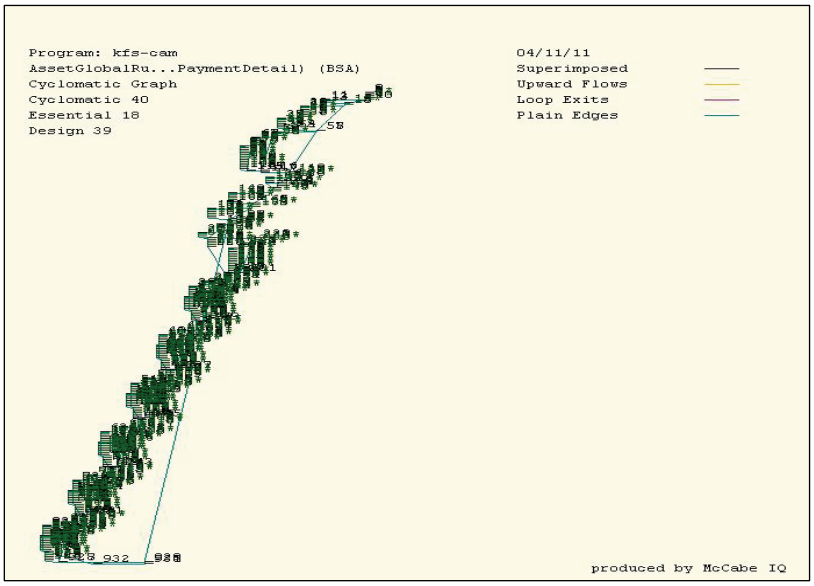


FIGURE 6 is the flowgraph of one of the most complex modules in kfs-cam. Visually, it is easy to see that this module is overly complex with high levels of dense print for the module's logic.

FIGURE 6
FLOWGRAPH OF AN OVERLY COMPLEX KFS-CAM MODULE WITH A CYCLOMATIC COMPLEXITY OF 40



Purchasing/Account Payable

The kfs-purap sub-system represents a mixture of old and new code. Most of the development was conducted at CSU to facilitate integration of SciQuest (an existing purchasing system) with Kuali. The kfs-purap application contains 547 source files. Parsing by McCabe IQ resulted in 5,778 modules. One hundred twenty-one of these modules had a v > 10. Thirty out of the 121 modules had a v > 20. Of these modules, several stood out as very complex, such as:

- Processing Taxes
- Processing Encumbrances
- Purchase Order Creation

Based on their cyclomatic complexity, the three functional areas noted above should be problematic.

Key profile indicators for kfs-purap are as follows:

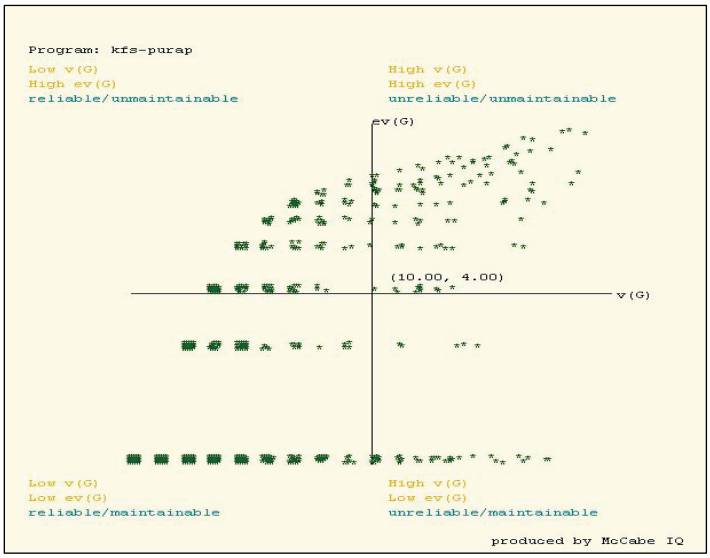
- The average cyclomatic complexity is 2.04.
- The average essential complexity is 1.37.
- The average module design complexity is 1.93.
- The total executable lines of code are 50,366.
- The cyclomatic complexity density is .2345.

A further review of the profile indicators for modules that had a v > 10 generate the following results:

- The average cyclomatic complexity is 17.9.
- The average essential complexity is 7.8.
- The average module design complexity is 16.2.
- The total executable lines of code are 10,435.
- The cyclomatic complexity density is .2090.

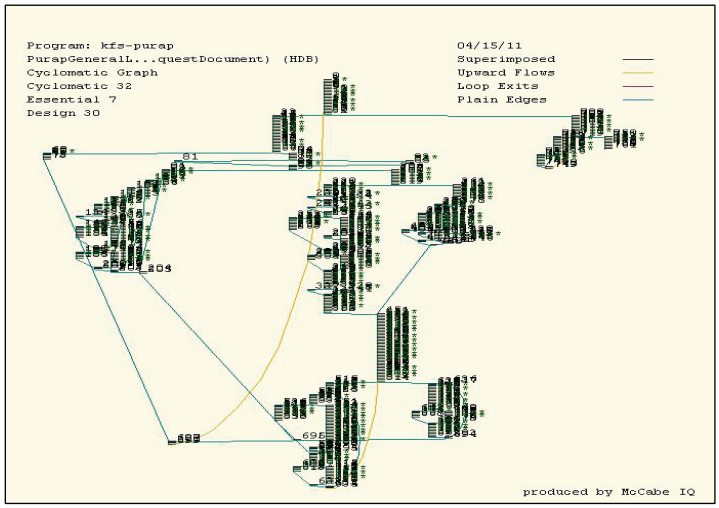
A simple visual comparison between the kfs-purap scatter plot, presented in FIGURE 7, and the other three application scatter plots shows that kfs-purap has noticeably more unreliable/un-maintainable modules (121 modules in quadrant one with v > 10) than the other three applications.

FIGURE 7
SCATTER PLOT WITH A NOTICEABLE INCREASE IN UNRELIABLE MODULES



Because of their high cyclomatic complexity numbers, Processing Taxes, Processing Encumbrances and Purchase Order Creation are high risk modules. As shown in FIGURE 8 which illustrates the module logic for Processing Encumbrances, these modules will likely prove to be problematic. Illustrated in this figure, Processing Encumbrances is a very high risk module with high cyclomatic complexity (v = 32), high essential complexity (ev = 7), and high module design complexity (iv = 30).

FIGURE 8
FLOWGRAPH OF A POTENTIALLY PROBLEMATIC PROCESSING ENCUMBRANCES MODULE



ANALYSIS

The McCabe IQ parsing described above results in a metric profile for each of the four applications tested. These profiles are summarized in TABLE 2. Each profile includes averages calculated based on all of the parsed modules for each application. The relatively low averages are a result of the large number of modules that fall below the McCabe cyclomatic complexity threshold of 10. (McCabe Staff) In this case, the term threshold means the value of a metric above which there is an element is of interest. Typically, below a threshold, metrics do not correlate with real effects, and code elements below threshold usually do not require code to be reviewed or modified. (McCabe Staff)

TABLE 2
APPLICATION PROFILE

Static Metrics	kfs-cab	kfs-bc	kfs-cam	kfs-purap
Number of Source Files	59	343	160	547
Number of Parsed Modules	868	6,266	2,221	5,778
Design Complexity, S0	1,780	8,565	3,874	11,166
Integration Complexity, S1	913	2,300	1,654	5,389
Total cyclomatic complexity	1,888	8,809	4,065	11,811
Average cyclomatic complexity	2.18	1.41	1.83	2.04
Maximum - cyclomatic complexity	18	53	40	78
Average essential complexity	1.40	1.11	1.26	1.37
Maximum - essential complexity	13	30	18	43
Total module design complexity	1,780	8,565	3,874	11,166
Average module integration complexity	2.05	1.37	1.74	1.93
Maximum - module design complexity	16	46	39	76
Total executable SLOC	7,477	44,445	16,774	50,366
Average executable SLOC	8.6	7.09	7.55	8.72
Maximum - executable SLOC	122	205	306	602
Density Total v(G) / Total executable SLOC	0.252508	0.1982	0.242339	0.234503
Reported Errors (closed & open)	8	3	86	153

The McCabe Risk Application Profile in TABLE 3 displays the profile of all of the high risk modules for each application (modules with a v > 10). This table accentuates the low quality of the kfs-purap code. Its average cyclomatic, essential, and module design complexities are higher than the other three applications. Based on the profile provided by TABLE 3, one could conclude that the least number of errors will be found in kfs-cab because its high risk modules have the lowest average v.

- Based on cyclomatic complexity, the quality of the applications from lowest to highest is: kfs-purap, kfs-cam, kfs-bc, kfs-cab.
- Based on essential complexity, the quality of the applications from lowest to highest is: kfs-purap, kfs-cab, kfs-bc, kfs-cam.
- Based on module integration complexity, the quality of the applications from lowest to highest is: kfs-purap, kfs-cam, kfs-bc, kfs-cab.
- Based on density, the quality of the applications from lowest to highest is: kfs-cab, kfs-cam, kfs-purap, kfs-bc.

TABLE 3
McCABE RISK APPLICATION PROFILE- HIGH RISK MODULES

Static Metrics	kfs-cab	kfs-bc	kfs-cam	kfs-purap
Number of Parsed Modules with v(G)>10	12	45	32	121
Total cyclomatic complexity-v(G)>10	153	696	523	2,181
Average cyclomatic complexity-v(G)>10	12.8	15.4	16.3	17.9
Maximum - cyclomatic complexity-v(G)>10	18	53	40	78
Total essential complexity-v(G)>10	92	270	180	951
Average essential complexity-v(G)>10	7.6	6.0	5.6	7.8
Maximum - essential complexity-v(G)>10	13	30	15	43
Total module design complexity-v(G) >10	131	582	454	1,961
Average module integration complexity-v(G) >10	11.2	14.0	14.5	16.2
Maximum - module design complexity-v(G) >10	16	46	39	76
Total executable SLOC-v(G)>10	551	3,764	2,450	10,435
Average executable SLOC-v(G)>10	45.9	83.6	76.6	85.7
Maximum - executable SLOC-v(G)>10	87	534	229	491
Density Total v(G) / Total lines of code-v(G)>10	.2778	.1849	.2135	.2090
Reported Errors (closed & open)	8	3	86	153

CONCLUSIONS

As articulated earlier, the objectives achieved by this project were a description of expanding role of Kuali enterprise software in universities and colleges and an analysis of the code quality produced by the consortium. Since it was conceived in 2004, The Kuali Foundation has successfully implemented six enterprise software modules for higher education on a timely, cost-effective manner including:

- Financial System (2005): Financial software that meets the needs of all Carnegie Class institutions
- Coeus (2006): Research administration for grants administration to federal funding agencies
- Student (2007): Business needs of students, faculty and institutions throughout the academic lifecycle
- Rice (2007): Middleware products that integrate products allowing applications to be built in an agile fashion
- Open Library Environment (2010): Integration of academic and research libraries for managing and delivering intellectual information
- Mobility (2011): Connect mobile devices to a variety of campus systems

Prior to this study, no research had been completed of the Kuali software analyzing the code quality. In this study four modules of the Financial System were analyzed for their quality attributes. McCabe IQ was used to determine the McCabe metrics including cyclomatic, essential, and module design complexities. Overall, the code quality was found to be good quality with about 98% of the software modules exhibiting cyclomatic complexities less than or equal to 10. Overall, the findings showed that risker software as a percentage of total software modules for the four studied modules is as follows:

- Budget Construction (BC) - 1.3%
- Capital Asset Builder (CAB) - 0.7%
- Capital Asset Management (CAM) - 1.4%
- Purchasing/Account Payable (PUR/AP) - 2.1%

The Kuali Foundation is serving the university community with enterprise software targeted specifically for university and college business models, and it is delivering cost-effective, quality processes to the higher education community.

REFERENCES

Aberdour, M. (2007). Achieving Quality in Open Source Software. *IEEE Software*, 58-64.

CURL Staff. (05, 14 2010). *MIDESS Functiona and Technical Requirements Specification.* Retrieved 2011, from http://ludos.leeds.ac.uk/midess/MIDESS%20workpackage%202%20-%20Functional%20and%20Technical%20Requirements%20Specification.pdf

Hanganu, G. (2011, 03 11). *OSS Watch Open Source Software Advisory Service.* Retrieved 2011, from www.oss-watch.ac.uk: http://www.oss-watch.ac.uk/resources/communitysource.xml

Jeffery, M. (n.d.). *Northwestern University.* Retrieved from http://www.kellogg.northwestern.edu/: http://www.kellogg.northwestern.edu/faculty/jeffery/htm/publication/ROIforITProjects.pdf

Kaur, K., Minhas, K., Mehan, N., & Kakkar, N. (2009). Static and Dynamic Complexity Analysis of Software Metrics. *World Academy of Science, Engineering and Technology*, 159-161.

Koha. (1999). *Koha*. Retrieved from http://koha.org/: http://koha.org/

Kuali. (2010). *Kuali - OLE*. Retrieved from www.kuali.org: http://www.kuali.org/ole

Kuali Foundation. (2010). *Kuali Foundation - Kuali Coeus (KC)*. Retrieved from www.kuali.org: http://www.kuali.org/kc

McCabe Staff. (n.d.). *McCabe Demostration Page*. Retrieved 2011, from www.mccabe.com: http://www.mccabe.com/contact_iqDemo.htm

McCabe Staff. (n.d.). *McCabe IQ Glossary of Terms*. Retrieved from www.mccabe.com: http://www.mccabe.com/iq_research_iqgloss.htm#t

McCabe Staff. (n.d.). *McCabe Metrics.* Retrieved 2011, from www.mccabe.com: http://www.mccabe.com/pdf/McCabe%20IQ%20Metrics.pdf

NASA IV&V Facility. (2008, 6 10). *Metrics Data Program - NASA IV&V Facility - Complexity Metrics*. Retrieved 2011, from www.nasa.gov: http://mdp.ivv.nasa.gov/complexity_metrics.html

NASA. (n.d.). *Overview of Software Reliability*. Retrieved 2011, from Goddard Space Flight Center: http://sw-assurance.gsfc.nasa.gov/disciplines/reliability/index.php

NLM Digital Repository Evaluation and Selection Working Group. (2008, 12 02). *Recommendations on NLM Digital Repository Software.* Retrieved 2011, from http://www.nlm.nih.gov/digitalrepository/DRESWG-Report.pdf

Shelly, G. B., & Harry, R. (2010). *Systems Analysis an Design.* Course Technology.

Stoneburner, W. (2010, 03 31). *SMERFS Software Reliability Overview*. Retrieved 2011, from www.slingcode.com: http://slingcode.com/smerfs/overview.php

Technical Council. (2007, 12). *Kuali Standards Version 1.2.* Retrieved 2011, from http://kuali.org: http://kuali.org/files/pdf/KualiStandards.pdf

Traylor, P. S. (2006, 02 13). *http://www.infoworld.com.* Retrieved 2011, from InfoWorld: http://www.infoworld.com/print/20676

Wheeler, B. (2007). Open Source 2010 Reflections on 2007. *EDUCAUSE Review*, 49-67.

Wheeler, B., & DeStefano, J. (n.d.). *Mitigating the Risks of Big Systems.* Retrieved from http://www.nacubo.org/: http://www.nacubo.org/Business_Officer_Magazine/Magazine_Archives/July-August_2007/Mitigating_the_Risks_of_Big_Systems.html

Wikipedia. (2010, 08 11). *Wikipedia - The Free Encyclopedia.* Retrieved 2011, from www.wikipedia.org: http://en.wikipedia.org/wiki/Community_source