Software Complexity Model

Thuc Tran
School of Engineering and Applied Science
The George Washington University
ttran21@gwu.edu

NDIA Systems Engineering Conference 2017

Software Complexity Model

What is Complexity?

*“not easy to understand or explain **:** not simple ”*

“having parts that go together in complicated ways”

“having many varied interrelated parts, patterns, or elements and consequently hard to understand”

What is Software Complexity?

Software that is *“not easy to understand or explain **:** not simple ”*

Software *“having parts that go together in complicated ways”*

Software *“having many varied interrelated parts, patterns, or elements and consequently hard to understand”*

Software Complexity makes software difficult to understand and support

Source: https://www.merriam-webster.com/dictionary/complex

Software Complexity Model

Problem Statement

The lack of a comprehensive software complexity measurement framework leads to an increase of over 90% in software maintenance cost.

Research Objective

The research aims to measure the complexity of software applications through a comprehensive analysis using different dimensions of characteristics. The result will be a score which comprehensively represents the dimensions of software complexity.

Source: Software Maintenance Costs (Koskinen, 2015)

Software Complexity Model

Impacts of Software Complexity

- More than 90% of overall software lifecycle cost can be devoted to maintenance

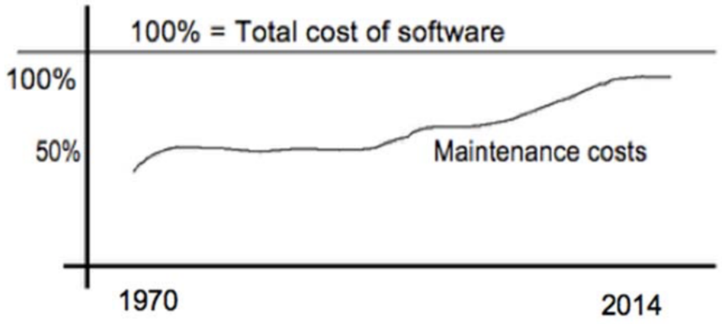


Figure 1: Development of Software maintenance costs as percentage of total cost

Year	Proportion of software maintenance costs	Definition	Reference
2000	>90%	Software cost devoted to system maintenance & evolution / total software costs	Erlikh (2000)
1993	75%	Software maintenance / information system budget (in Fortune 1000 companies)	Eastwood (1993)
1990	>90%	Software cost devoted to system maintenance & evolution / total software costs	Moad (1990)
1990	60-70%	Software maintenance / total management information systems (MIS) operating budgets	Huff (1990)
1988	60-70%	Software maintenance / total management information systems (MIS) operating budgets	Port (1988)
1984	65-75%	Effort spent on software maintenance / total available software engineering effort.	McKee (1984)
1981	>50%	Staff time spent on maintenance / total time (in 487 organizations)	Lientz & Swanson (1981)
1979	67%	Maintenance costs / total software costs	Zelkowitz *et al.* (1979)

Source: Software Maintenance Costs (Koskinen, 2015)
How to save on software maintenance costs (Vries & Burki, 2014)

Software Complexity Model

Impacts of Software Complexity

- Analysis of software accounts for nearly 50% of maintenance development

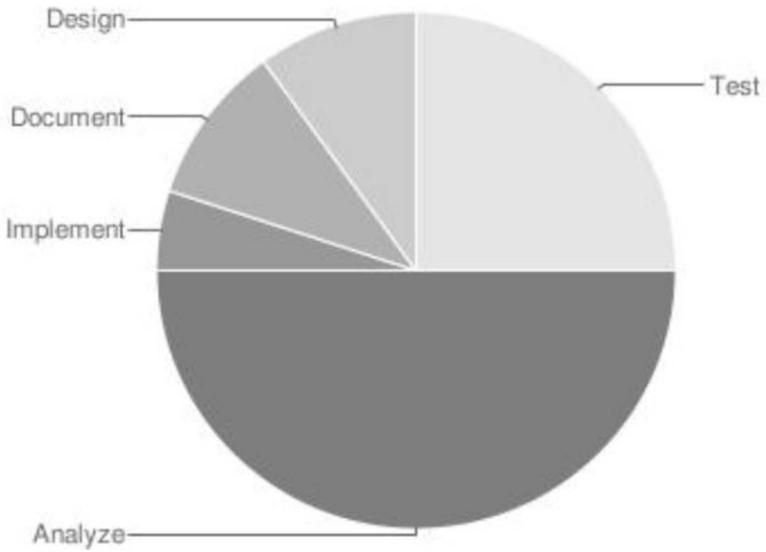
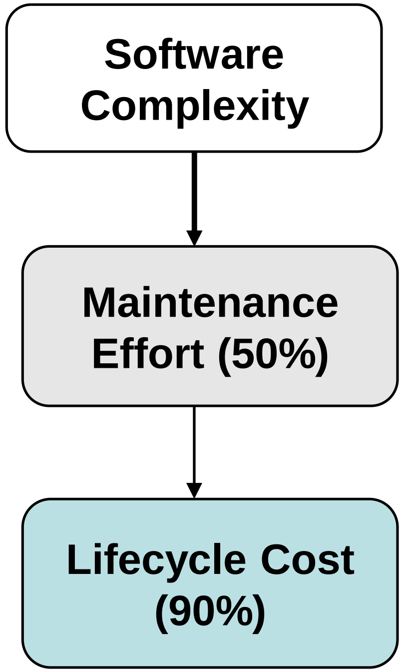


Figure 2: Analyzing is almost 50% of the total maintenance effort



Source: Software Development Practices, Software Complexity, and Software Maintenance (Banker et al, 1998)
How to save on software maintenance costs (Vries & Burki, 2014)

Software Complexity Model

Software Product Quality Model – ISO/IEC 9126 (2001)

- **Functionality –** The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
- **Reliability –** The capability of the software product to maintain a specified level of performance when used under specified conditions.
- **Usability –** The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.
- **Efficiency –** The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
- **Maintainability –** The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
- **Portability –** The capability of the software product to be transferred from one environment to another.

Source: ISO/IEC 9126

Software Complexity Model

Software Product Quality Model – ISO/IEC 9126 (2001)

Dimension	Sub-Dimension	Definition
Functionality	Suitability	• The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.
	Accuracy	• The capability of the software product to provide the right or agreed results or effects with the needed degree of precision.
	Interoperability	• The capability of the software product to interact with one or more specified systems.
	Security	• The capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.
	Functionality Compliance	• The capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality.
Reliability	Maturity	• The capability of the software product to avoid failure as a result of faults in the software.
	Fault Tolerance	• The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
	Recoverability	• The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure.
	Reliability Compliance	• The capability of the software product to adhere to standards, conventions or regulations relating to reliability.
Usability	Understandability	• The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
	Learnability	• The capability of the software product to enable the user to learn its application.
	Operability	• The capability of the software product to enable the user to operate and control it.
	Attractiveness	• The capability of the software product to be attractive to the user.
	Usability Compliance	• The capability of the software product to adhere to standards, conventions, style guides or regulations relating to usability.

Software Complexity Model

Software Product Quality Model – ISO/IEC 9126 (2001)

Dimension	Sub-Dimension	Definition
Efficiency	Time Behavior	• The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.
	Resource Utilization	• The capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions.
	Efficiency Compliance	• The capability of the software product to adhere to standards or conventions relating to efficiency.
Maintainability	Analyzability	• The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.
	Changeability	• The capability of the software product to enable a specified modification to be implemented.
	Stability	• The capability of the software product to avoid unexpected effects from modifications of the software.
	Testability	• The capability of the software product to enable modified software to be validated.
	Maintainability Compliance	• The capability of the software product to adhere to standards or conventions relating to maintainability.
Portability	Adaptability	• The capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered.
	Installability	• The capability of the software product to be installed in a specified environment.
	Co-Existence	• The capability of the software product to co-exist with other independent software in a common environment sharing common resources.
	Replaceability	• The capability of the software product to be used in place of another specified software product for the same purpose in the same environment.
	Portability Compliance	• The capability of the software product to adhere to standards or conventions relating to portability.

Software Complexity Model

Software Product Quality Model – ISO/IEC 9126 (2001)

- Compliance is a part of every dimension and can be considered a dimension on its own
- *Note: The following displays all attributes from the ISO/IEC 9126 Product Quality Model, but not all dimensions / sub-dimensions will be used:*

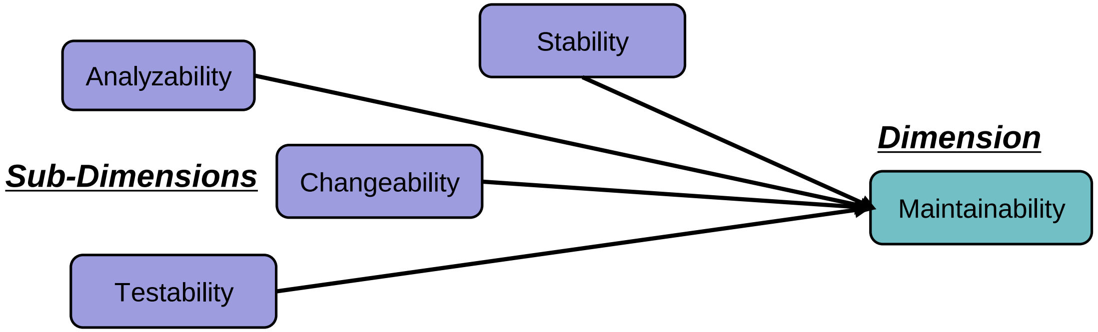
Dimensions	Sub-Dimensions
Functionality	• Suitability • Accuracy • Interoperability • Security • Functionality Compliance
Reliability	• Maturity • Fault Tolerance • Recoverability • Reliability Compliance
Usability	• Understandability • Learnability • Operability • Attractiveness • Usability Compliance
Efficiency	• Time Behavior • Resource Utilization • Efficiency Compliance
Maintainability	• Analyzability • Changeability • Stability • Testability • Maintainability Compliance
Portability	• Adaptability • Installability • Co-Existence • Replaceability • Portability Compliance

Dimensions	Sub-Dimensions
Functionality	• Suitability • Accuracy • Interoperability • Security
Reliability	• Maturity • Fault Tolerance • Recoverability
Usability	• Understandability • Learnability • Operability • Attractiveness
Efficiency	• Time Behavior • Resource Utilization
Maintainability	• Analyzability • Changeability • Stability • Testability
Portability	• Adaptability • Installability • Co-Existence • Replaceability
Compliance	• Functionality Compliance • Reliability Compliance • Usability Compliance • Efficiency Compliance • Maintainability Compliance • Portability Compliance

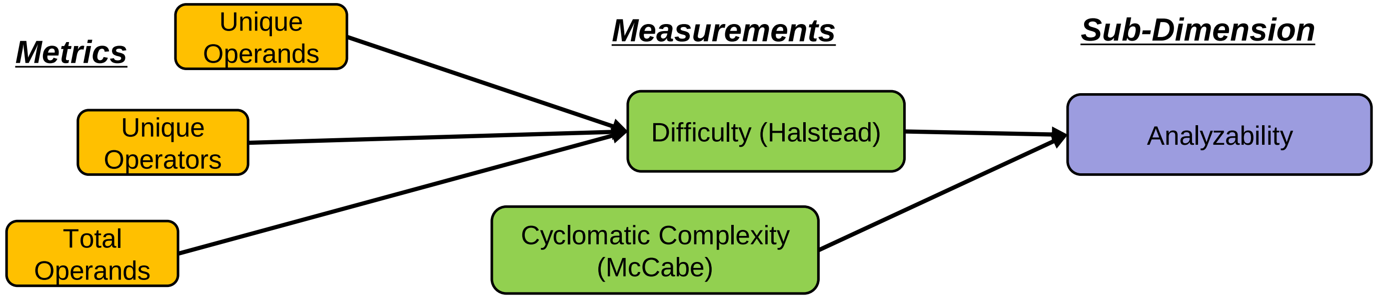
Software Complexity Model

Software Product Quality Model – ISO/IEC 9126 (2001)

- **Dimensions** are comprised of **Sub-Dimensions**



- **Sub-Dimensions** are comprised of various **measurements**
- **Measurements** may use many different **metrics**



Software Complexity Model

Software Metrics

- Software Metrics identify a ***value*** that represents a characteristic of the software
- Software Metrics contribute to the evaluation of Software Measurements

Metric Category	Metric Type	Metric
Complexity	Size	• Lines of Code
	Interface Complexity	• Number of Attributes and Methods
		• Number of Local Methods
	Structural Complexity	• McCabe Cyclomatic Complexity
		• Weighted Method Count
		• Response for a Class

Source: ARISA Compendium of Software Quality Standards and Metrics - Version 1.0

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Software Complexity Model

Software Metrics

Metric Category	Metric Type	Metric
Architecture and Structure	Inheritance	• Depth of Inheritance Tree
		• Number of Children
	Coupling	• Afferent Coupling
		• Coupling Between Objects
		• Change Dependency Between Classes
		• Change Dependency of Classes
		• Efferent Coupling
		• Coupling Factor
		• Data Abstraction Coupling
		• Instability
		• Locality of Data
		• Message Passing Coupling
		• Package Data Abstraction Coupling
	Cohesion	• Lack of Cohesion in Methods
		• Improvement of LCOM
		• Tight Class Cohesion

Software Complexity Model

Software Metrics

Metric Category	Metric Type	Metric
Design Guidelines and Code Conventions	Documentation	• Lack of Documentation
	Code Conventions	

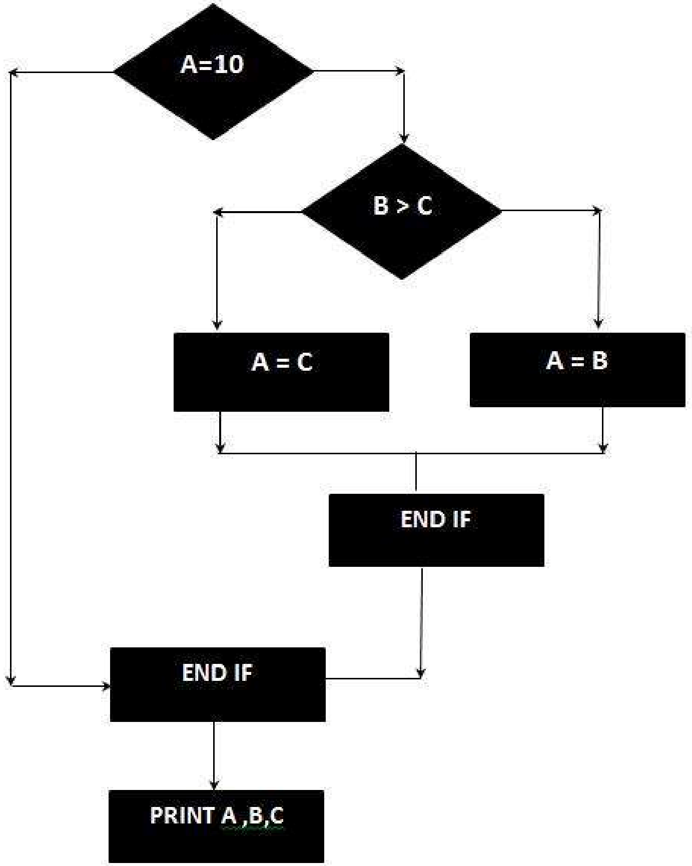
Source: ARISA Compendium of Software Quality Standards and Metrics - Version 1.0

Software Complexity Model

Cylcomatic Complexity

Example Complexity:

```
IF A = 10 THEN
 IF B > C THEN
   A = B
 ELSE
   A = C
 ENDIF
ENDIF
Print A
Print B
Print C
```



v(G) = e – n + p

- **v(G)** = cyclomatic number
- **e** = edges
- **n** = nodes
- **p** = connected components

v(G) = 8 – 7 + 2 = 3

- **e** = 8
- **n** = 7
- **p** = 2

Source: https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm

Software Complexity Model

Software Science Metrics

```
void sort ( int *a, int n ) {
    int i, j, t;
    if (n<2) return;
    for (i=0; i<n-1; i++) {
        for (j=i+1; j<n; j++) {
            if (a[i] > a[j]) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}
```

Operators			
<	3	**{**	3
=	5	**}**	3
>	1	**+**	1
-	1	**++**	2
,	2	**for**	2
;	9	**if**	2
(	4	**int**	1
)	4	**return**	1
[]	6		

Operands	
0	1
1	2
2	1
a	6
i	8
j	7
n	3
t	3

Source: http://www.win.tue.nl/~aserebre/2IS55/2011-2012/10.pdf

Software Complexity Model

Software Science Metrics

	Total	Unique
Operators	N1 = 50	n1 = 17
Operands	N2 = 30	n2 = 7

n1 = unique operators
n2 = unique operands
N1 = total operators
N2 = total operands

Program Length (N) = N1 + N2
Vocabulary Size (n) = n1 + n2
Volume (V) = N * log2(n)
Difficulty (D) = (n1 / 2) * (N2 / n2)
Level (L) = 1 / D
Effort = D * VOL
Time (T) = E / 18
Bugs (B) = V / 3000

n1 = 17
n2 = 7
N1 = 50
N2 = 30

Program Length (N) = 80
Vocabulary Size (n) = 24
Volume (V) ≈ 392
Difficulty (D) ≈ 36
Level (L) ≈ 0.027
Effort = 14112
Time (T) = 784 (s)
Bugs (B) ≈ 0.1306

Source: http://www.win.tue.nl/~aserebre/2IS55/2011-2012/10.pdf

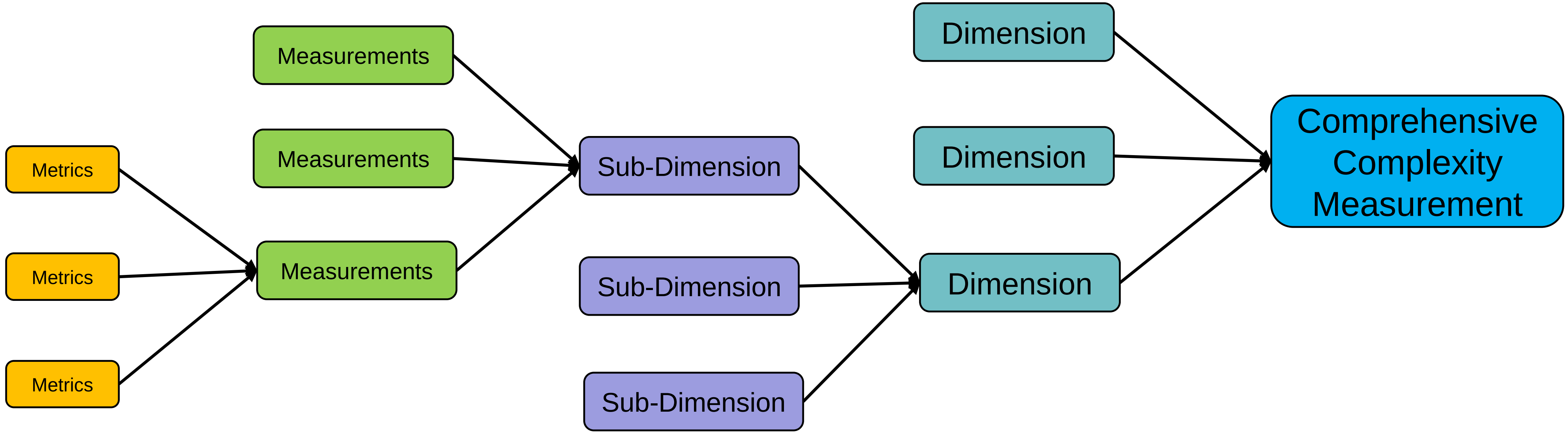
Software Complexity Model

Comprehensive Complexity Measurement

- Software Metrics identify a value that represents a characteristic of the software
- Metrics are used to calculate Software Measurements
- Software Measurements are used to evaluate Sub-Dimensions
- Sub-Dimensions are then used to evaluate Dimensions
- Dimensions can then be used to calculate a Comprehensive Complexity Measurement

Software Complexity Model

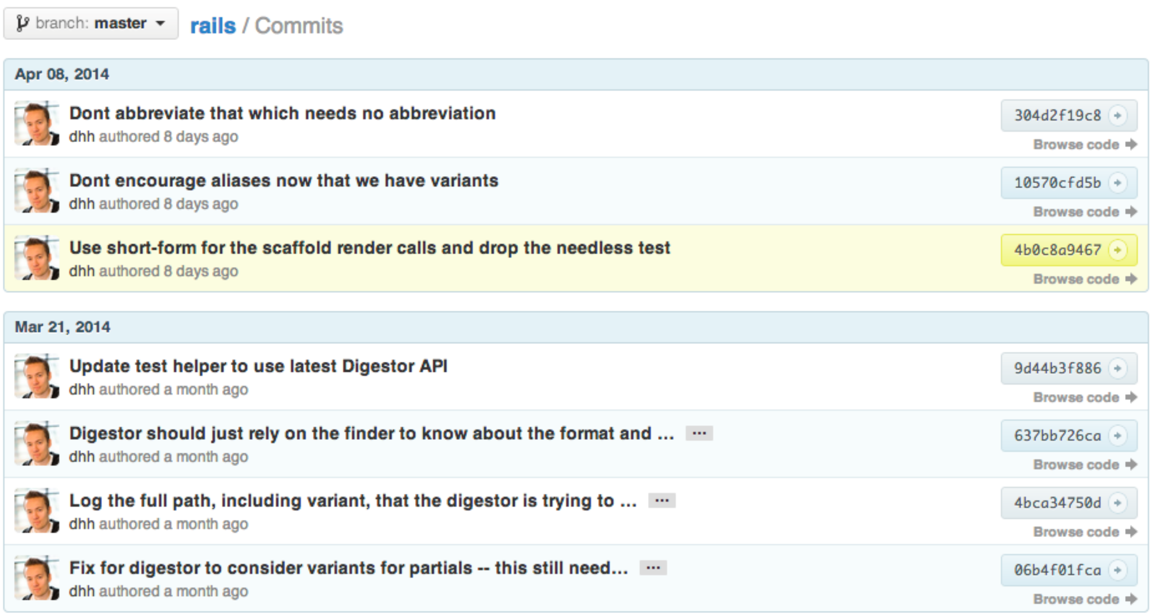
Comprehensive Complexity Measurement

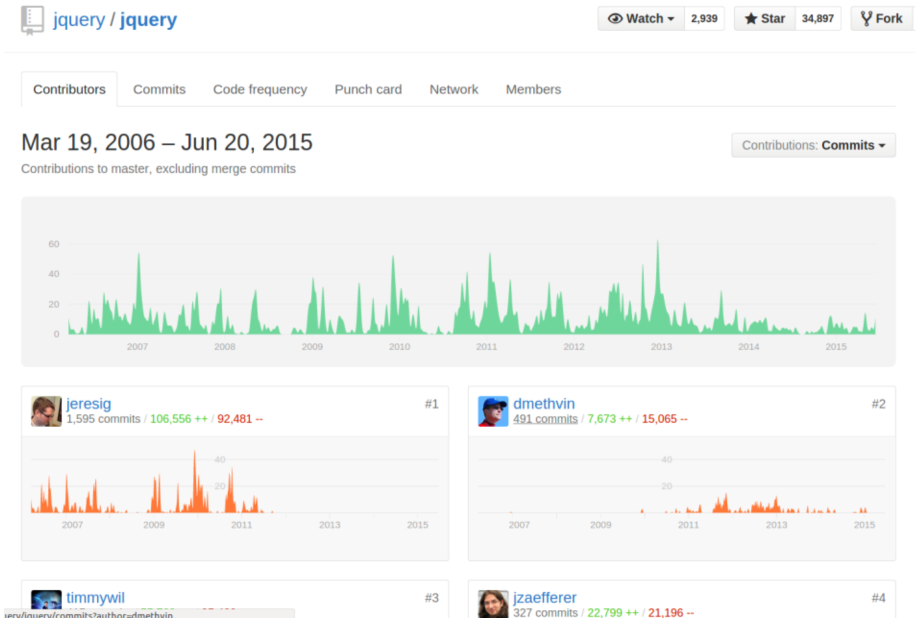


Software Complexity Model

Implementation

- Now we have a current score and a desired score, **so what?**
- The framework can then **recommend** changes that most significantly reduce the delta score; bringing the **current system** closer to the **most optimal system**
- **This can eventually be operationalized with a system like GitHub, a version control system that tracks changes *over time***





Questions


a.length&&(x=a[i])&&x.oSrc;i
d.MM_p=new Array();
arguments; for(i=0;
Image; d.MM_p[j++]
indexOf("?"))>0&&parent.
document; n=n.substring(0,p);}
i<d.forms.length;i++)
x=MM_findObj(n,d.layers
(n); return x;}
document.MM_sr=new Array;

References

- Scalet et al., 2000: ISO/IEC 9126
- Carlson, A. (n.d.). *University of Washington*. Retrieved 6 21, 2017, from Paul G. Allen School of Computer Science and Engineering: http://courses.cs.washington.edu/courses/cse403/96sp/coupling-cohesion.html
- Chidamber, S. R., & Kemerer, C. F. (1994, June). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering, 20*(6), 476-493.
- *Cyclomatic Complexity*. (n.d.). Retrieved 6 21, 2017, from tutorialspoint: https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm
- Halstead, M. (1977). *Elements of Software Science.* New York, NY: Elsevier.
- Holzmann, G. (2007, December). Conquering Complexity. 111-113.
- *https://www.merriam-webster.com*. (2017, 6 21). Retrieved from Merriam-Webster: https://www.merriam-webster.com/dictionary/complex
- Kafura, D., & Henry, S. (1981, September). Software Structure Metrics Based on Information Flow. *IEEE Transactions on Software Engineering, SE-7*(5), 510-518.
- McCabe, T. J. (1976, December). A Complexity Measure. *IEEE Transactions on Software Engineering, SE-2*(4), 308-320.
- *Measurement of Halstead Metrics with Testwell CMT++ and CMTJava (Complexity Measures Tool)*. (n.d.). Retrieved 6 21, 2017, from verifysoft: http://www.verifysoft.com/en_halstead_metrics.html
- Misra, S., Akman, I., & Colomo-Palacios, R. (2011). Framework for evaluation and validation of software complexity measures.
- Ortu, M., Destefanis, G., Murgia, A., Marchesi, M., Tonelli, R., & Adams, B. (n.d.). The JIRA Repository Dataset: Understanding Aspects of Software Development.
- Serebrenik, A. (2017, 6 21). Software Metrics. *Software Evolution*.
- Shao, J., & Wang, Y. (2003). A new measure of software complexity based on cognitive weights. *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering*.
- Stevens, W., Myers, G., & Constantine, L. (1974, June). Structured Design. *IBM Systems Journal, 13*(2), 115-139.
- *The Halstead Metrics*. (n.d.). Retrieved 6 21, 2017, from Virtual Machinery: http://www.virtualmachinery.com/sidebar2.htm
- Weyuker, E. (1988, September). Evaluating Software Complexity Measures. *IEEE Transactions on Software Engineering, 14*(9), 1357-1365.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

References

- Debbarma, M. K., Debbarma, S., Debbarma, N., Chakma, K., & Jamatia, A. (2013). A Review and Analysis of Software Complexity Metrics in Structural Testing. *International Journal of Computer and Communication Engineering*, *2*(2), 129–133. https://doi.org/10.7763/IJCCE.2013.V2.154
- Dehaghani, S., & Hajrahimi, N. (2013). Which factors affect software projects maintenance cost more. *Acta Informatica Medica*, *21*(October 2012), 63–66. https://doi.org/10.5455/aim.2012.21.63-66
- Gui, & Scott, P. D. (2008). New coupling and cohesion metrics for evaluation of software component reusability. *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008*, 1181–1186. https://doi.org/10.1109/ICYCS.2008.270
- Holvitie, J., & Leppänen, V. (2014). Illustrating software modifiability - Capturing cohesion and coupling in a force-optimized graph. *Proceedings - 2014 IEEE International Conference on Computer and Information Technology, CIT 2014*, 226–233. https://doi.org/10.1109/CIT.2014.112
- Husein, S., & Oxley, A. (2009). A coupling and cohesion metrics suite for object-oriented software. *ICCTD 2009 - 2009 International Conference on Computer Technology and Development*, *1*, 421–425. https://doi.org/10.1109/ICCTD.2009.209
- Kafura, D., & Reddy, G. R. (1987). The Use of Software Complexity Metrics in Software Maintenance. *IEEE Transactions on Software Engineering*, *SE-13*(3), 335–343. https://doi.org/10.1109/TSE.1987.233164
- Klemola, T., & Rilling, J. (2003). A Cognitive Complexity Metric Based on Category Learning. In *The Second IEEE International Conference on Cognitive Informatics, 2003. Proceedings.* (pp. 106–112).
- Kushwaha, D. S., & Misra, A. K. (2006). Improved cognitive information complexity measure: a metric that establishes program comprehension effort. *ACM SIGSOFT Software Engineering Notes*, *31*(5), 1–7. https://doi.org/10.1145/1163514.1163533
- Allen, E. B., Khoshgoftaar, T. M., & Chen, Y. (2001). Measuring coupling and cohesion of software modules: an information-theory approach. *Proceedings Seventh International Software Metrics Symposium*, (561), 124–134.
- Keshavarz, G., Modiri, N., & Pedram, M. (2011). A Model for the Controlled Development of Software Complexity Impacts. *International Journal of Computer Science and Information Security, 9*(6).
- Mancoridis, S., Mitchell, B. S., & Rorres, C. (1998). Using Automatic Clustering to Produce High-Level System Organizations of Source Code. *Program Comprehension, 1998. IWPC '98. Proceedings., 6th International Workshop*.
- Mitchell, B. S., & Mancoridis, S. (2006). On the automatic modularization of software systems using the Bunchtool. *IEEE Transactions on Software Engineering*, *32*(3), 193–208.
- Yau, S. S., & Collofello, J. S. (1980). Some Stability Measures for Software Maintenance. *IEEE Transactions on Software Engineering*, *SE-6*(6), 545–552.