Tenure Lecture Presentation

Sonia M. Pascua
Faculty
School of Library and Information Studies

Biography

Sonia M. Pascua

Faculty

School of Library and Information Studies

Academic Background

B. S. in Library and Information Science, School of Library and Information Science University of the Philippines – UP Diliman

Academic Background

Master of Science in Computer Science, Department of Computer Science, College of Engineering University of the Philippines – UP Diliman. (June 26, 2016)

Academic Background

Certified Librarian – Board for Librarian (November 2007)

One Year Full-Time Certificate in Information Technology, Specialization: Applications Development, University of the Philippines Information Technology Training Center – UP Diliman

- UPITTC Scholar

Professional Experiences in the Academe

Faculty, University of the Philippines School of Library and Information Studies (UP SLIS), June 2013 – Present

QMS Lead, eUP Project, March 2016 - December 2016

Program Developer Associate, eUP Project, June 2015 - July 2017

Project Manager, UPITDC – CHED Project, January 7, 2013 – March 7, 2013

Project Manager, UP ITDC - Technical and Professional Services for the Rehabilitation of CHED's Internet, Website and Email Services, April 10 – 27, 2012

Software Analyst, UP ITDC - CATT CSharpMigration_ChartGui Project of Fujitsu Ten Solutions Philippines, UP ITTC, February 2012 - April 2012

Project Manager, Youth Conference in IT (YCIT) 2010. February 17 and 18 2010

Professional Experiences in the Academe

Sr. Applications Development Training Officer, University of the Philippines IT Training Center (UPITTC), February 2010 – May 2013

Deputy Training Manager, University of the Philippines IT Training Center (UPITTC), February 2009 – May 2013

Deputy Quality Management Representative, ISO 9001-2008 QMS, University of the Philippines IT Training Center, February 2010 – April 2013

Speaker and Trainer

- **Software Quality Assurance**
- **Systems Thinking**
- **Project Management**
- **Embedded Indexing**
- **Understanding Digital Preservation**
- **Cloud Computing: Web Technologies**
- **Careers in IT**

Professional Experiences in the Corporate Industry

Faculty, University of the Philippines School of Library and Information Studies (UP SLIS), June 2013 – Present

IT Project Manager, Telus House McKinley West Buildout Telus International Philippines, January 2016 - Present

QMS Lead, eUP Project, March 2016 - December 2016

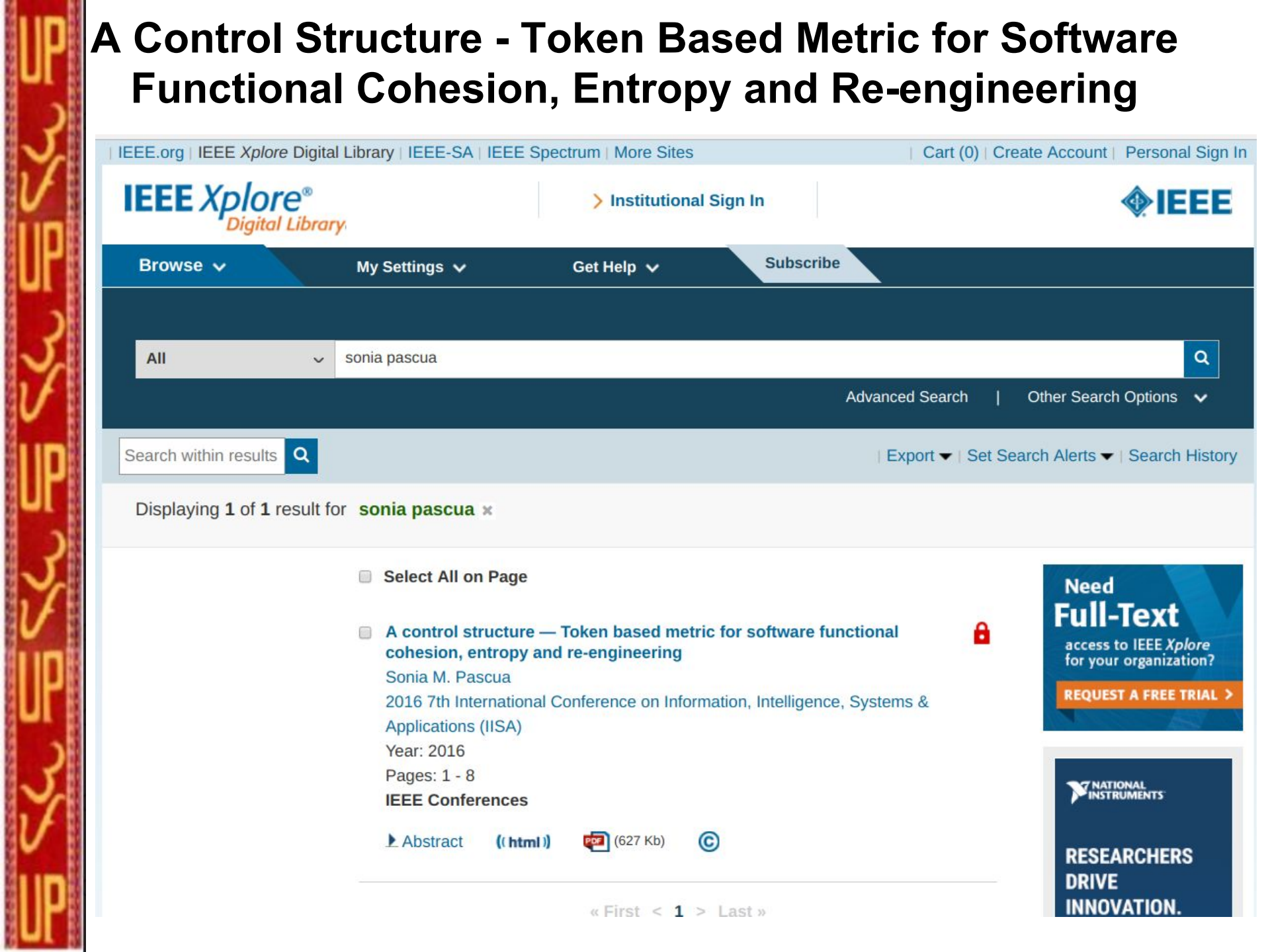
Program Developer Associate, eUP Project, June 2015 - July 2017

Project Manager, UPITDC – CHED Project, January 7, 2013 – March 7, 2013

Project Manager, UP ITDC - Technical and Professional Services for the Rehabilitation of CHED's Internet, Website and Email Services, April 10 – 27, 2012

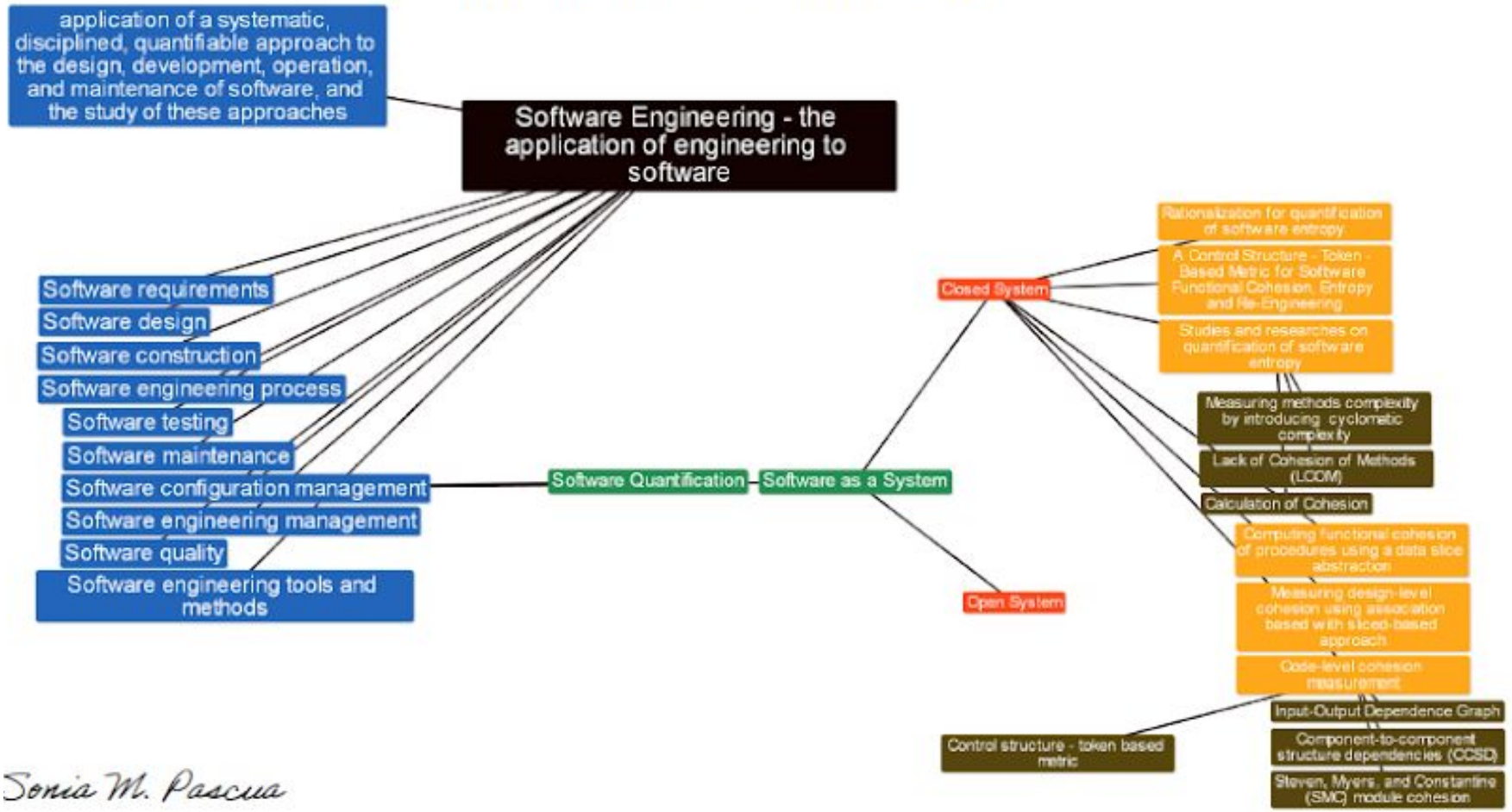
Software Analyst, UP ITDC - CATT CSharpMigration_ChartGui Project of Fujitsu Ten Solutions Philippines, UP ITTC, February 2012 - April 2012

A Control Structure - Token Based Metric for Software Functional Cohesion, Entropy and Re-engineering



A Control Structure - Token Based Metric for Software Functional Cohesion, Entropy and Re-engineering

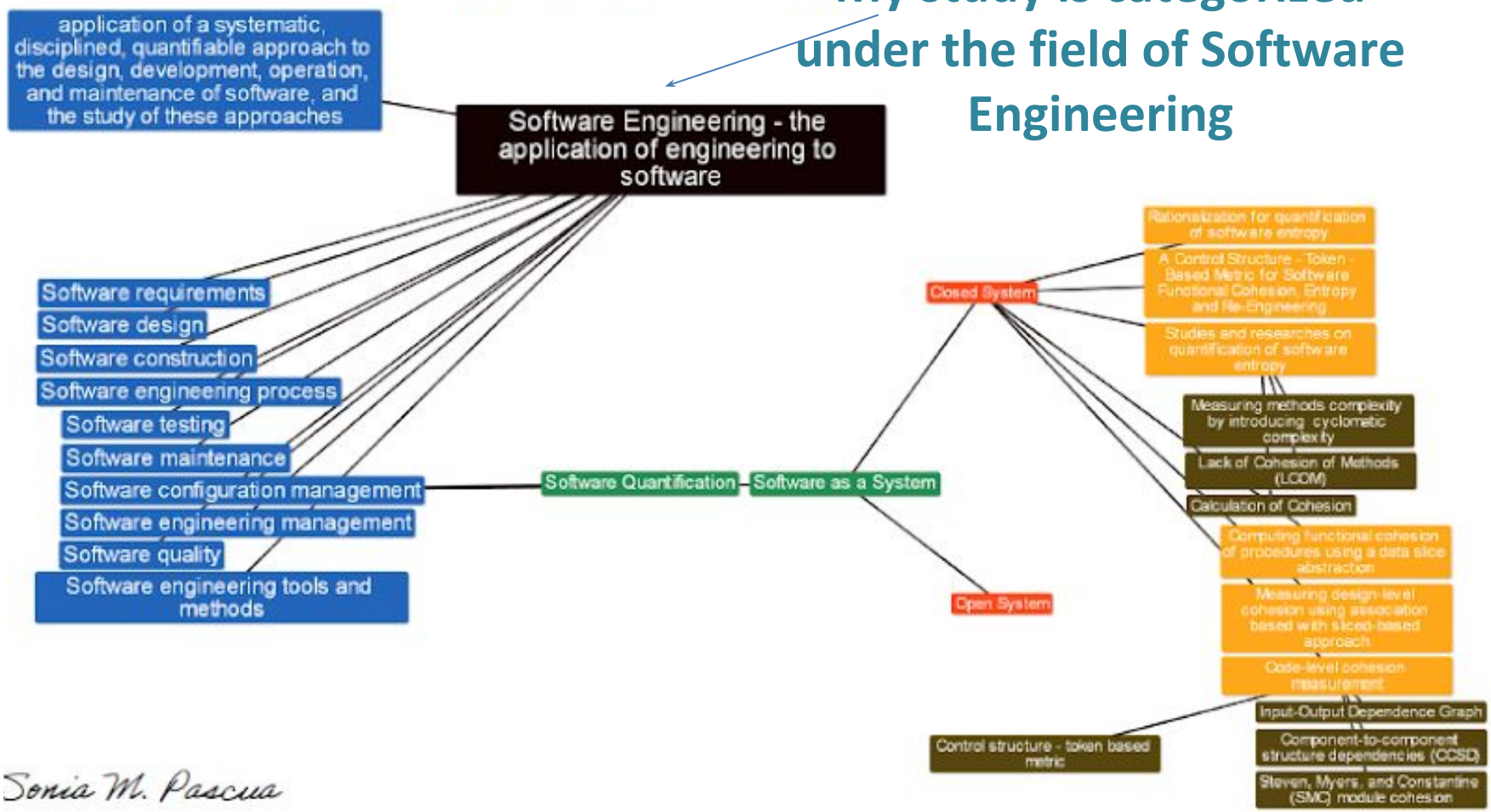
Theoretical Framework



Sonia M. Pascua

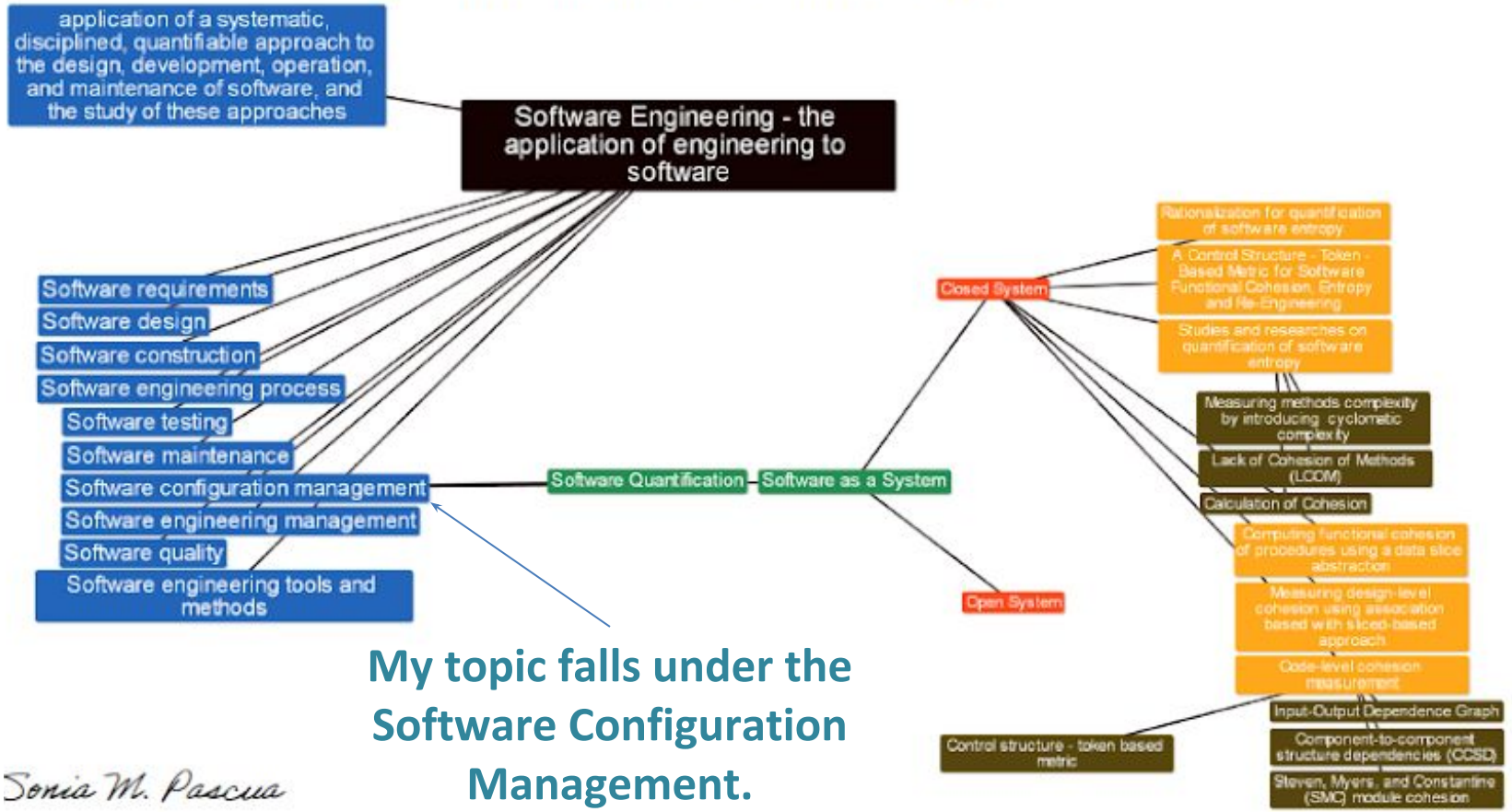
Theoretical Framework

My study is categorized under the field of Software Engineering



Sonia M. Pascua

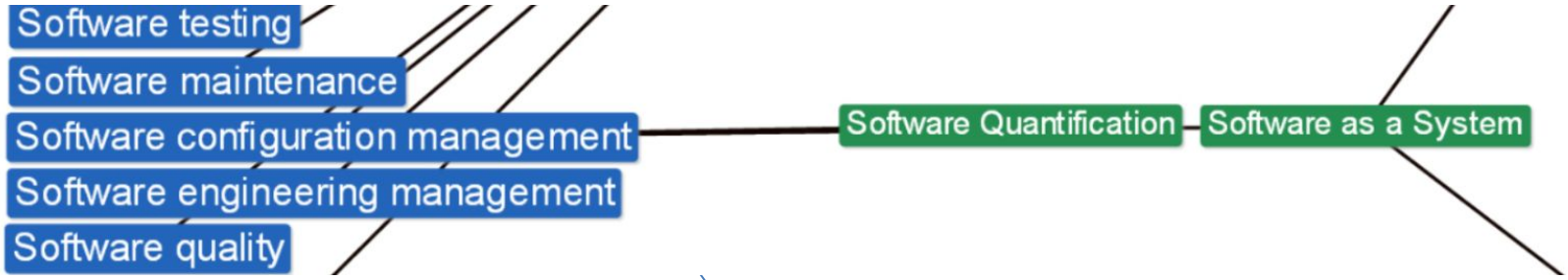
Theoretical Framework



My topic falls under the Software Configuration Management.

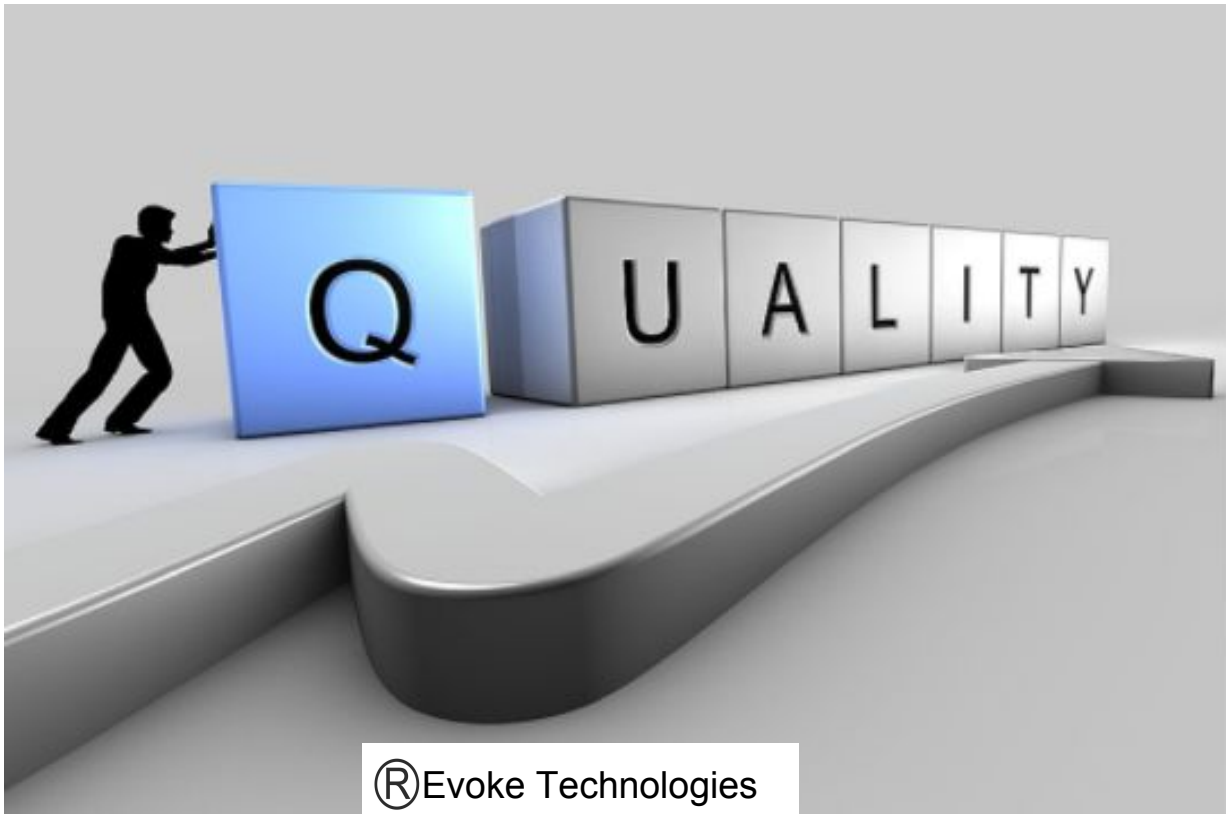
Sonia M. Pascua

Theoretical Framework



In software engineering, software configuration management (SCM) is the task of tracking and controlling changes in the software

Theoretical Framework



A field under Software Quality Assurance (SQA) which is task to measure and maintain the quality of the software.

Background of the Study

- When companies use technology such as software in process control and management, two of many possible objectives for continuous growth they have are

(1) efficiency in resource utilization and productivity, and

(2) an overhead amount of time and related cost in updating and managing this technology.

Background of the Study

- When companies use technology such as software in process control and management, two of many possible objectives for continuous growth they have are

(1) efficiency in resource utilization and productivity, and

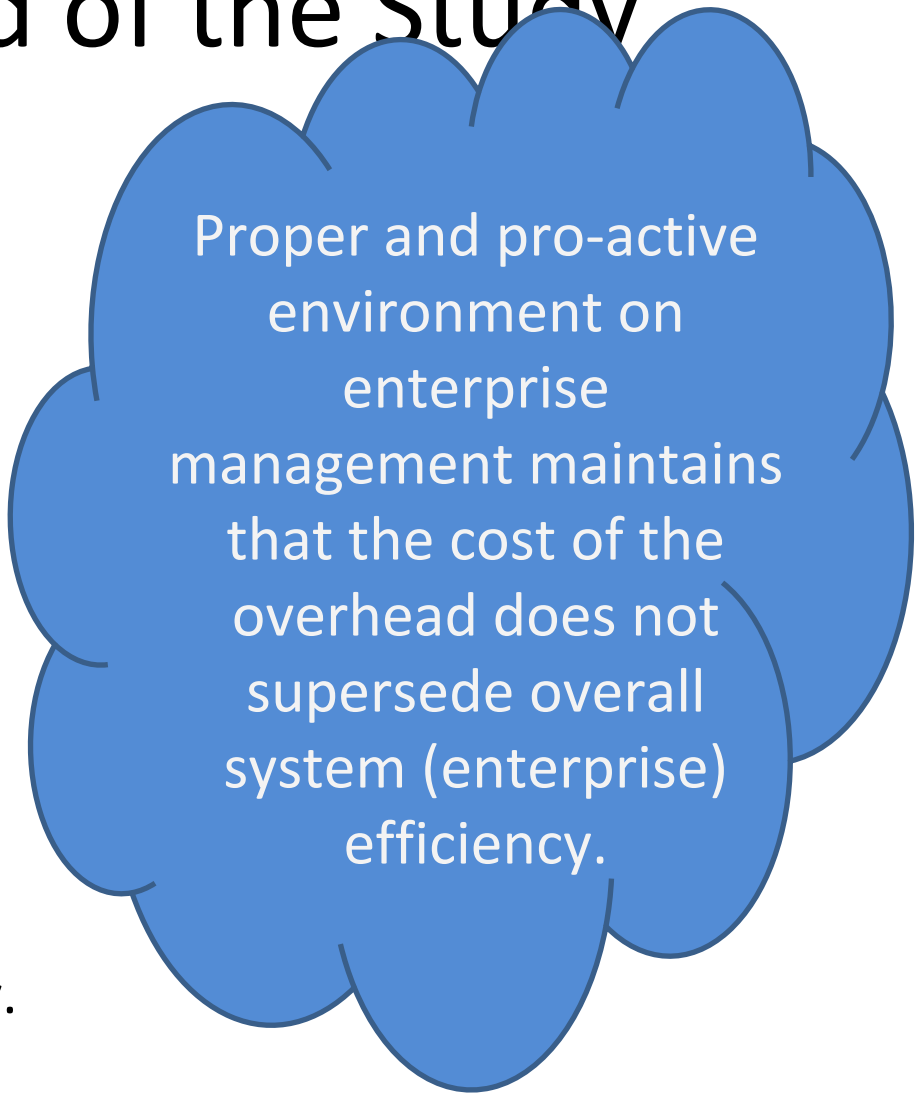
(2) an overhead amount of time and related cost in updating and managing this technology.

Background of the Study

- When companies use technology such as software in process control and management, two of many possible objectives for continuous growth they have are

(1) efficiency in resource utilization and productivity, and

(2) an overhead amount of time and related cost in updating and managing this technology.

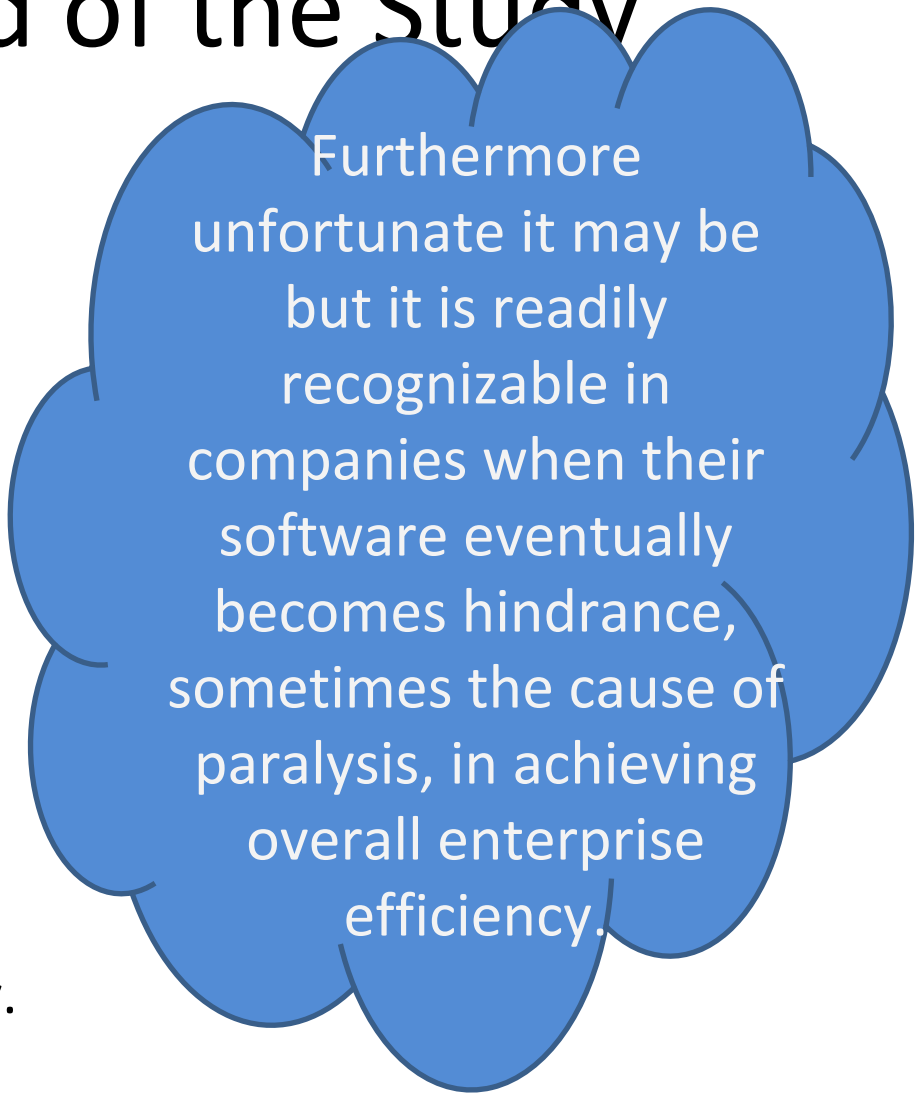


Background of the Study

- When companies use technology such as software in process control and management, two of many possible objectives for continuous growth they have are

(1) efficiency in resource utilization and productivity, and

(2) an overhead amount of time and related cost in updating and managing this technology.



Introduction

A work on software engineering by IVAR JOCOBSON describes software entropy as follows…

The second law of thermodynamics, in principle, states that a closed system's disorder cannot be reduced, it can only remain unchanged or increased.

A measure of this disorder is **entropy**.

Introduction

A work on software engineering by IVAR JOCOBSON describes software entropy as follows…

The second law of thermodynamics, in principle, states that a closed system's disorder cannot be reduced, it can only remain unchanged or increased.

A measure of this disorder is **entropy**.

This law also seems plausible for software systems;

as a system is modified, its disorder, or entropy, always increases.

This is known as **software entropy**.

Topic

This paper proposes a quantification of software entropy in terms of measuring control structures to variables cohesion.

Introduction

Within software development, there are similar theories;

Lehman (1985) suggested a number of laws, of which aid in the quantification of software entropy

- A computer program that is used will be modified
- When a program is modified, its complexity will increase, provided that one does not actively work against this.

The process of **code refactoring** can result in stepwise reductions in software entropy.

Introduction

Within software development, there are similar theories;

Lehman (1985) suggested a number of laws, of which aid in the quantification of software entropy

- A computer program that is used will be modified
- When a program is modified, its complexity will increase, provided that one does not actively work against this.

The process of **code refactoring** can result in stepwise reductions in software entropy.

Topic

This paper proposes a quantification of software entropy in terms of measuring control structures to variables **cohesion.**

Cohesion

- is the measure of the strength of functional association of elements within a procedure

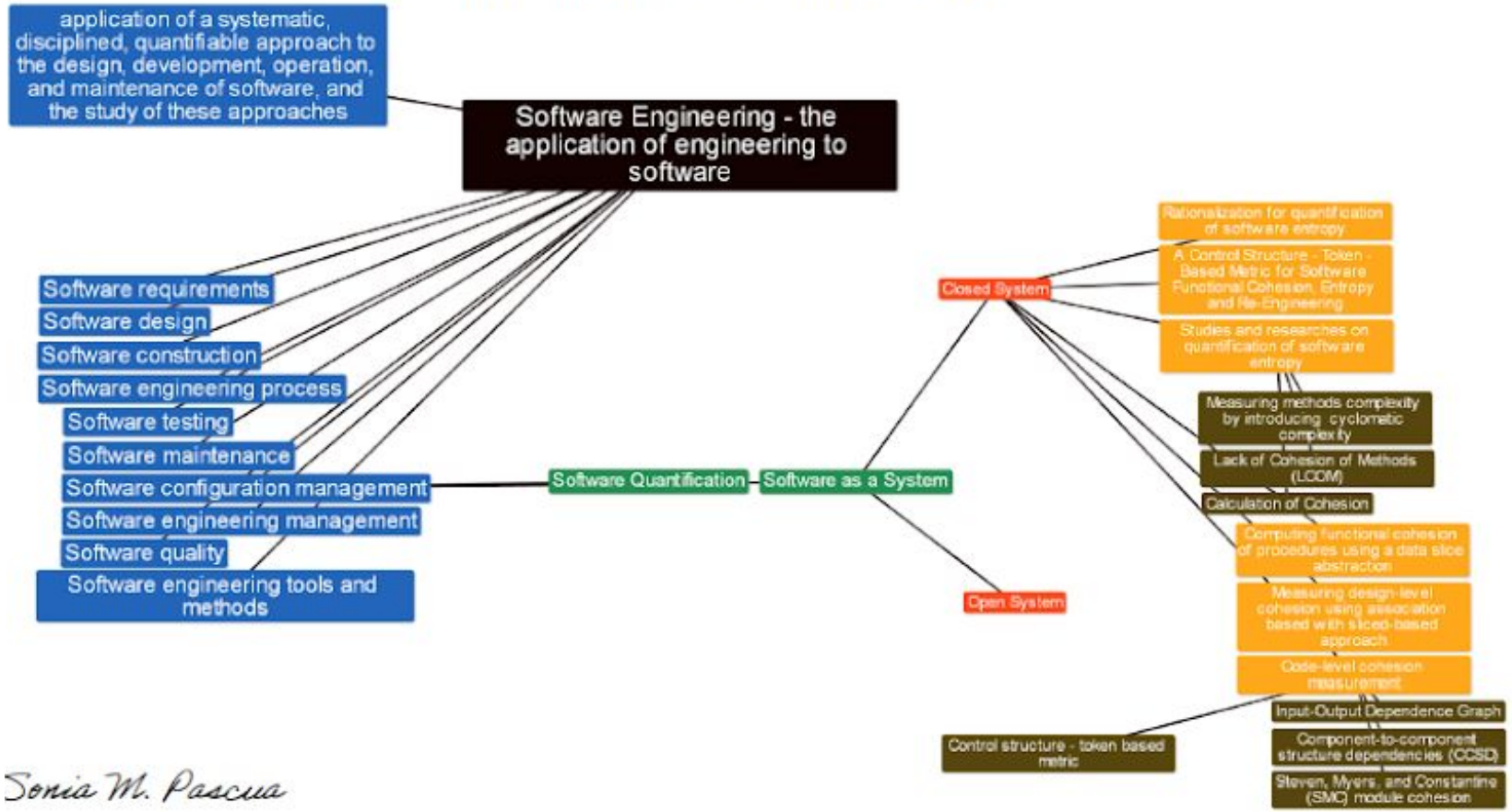
We strive for the highest possible cohesion

– i.e. make elements of a procedure as strongly related to one another as possible source

Why High Cohesion

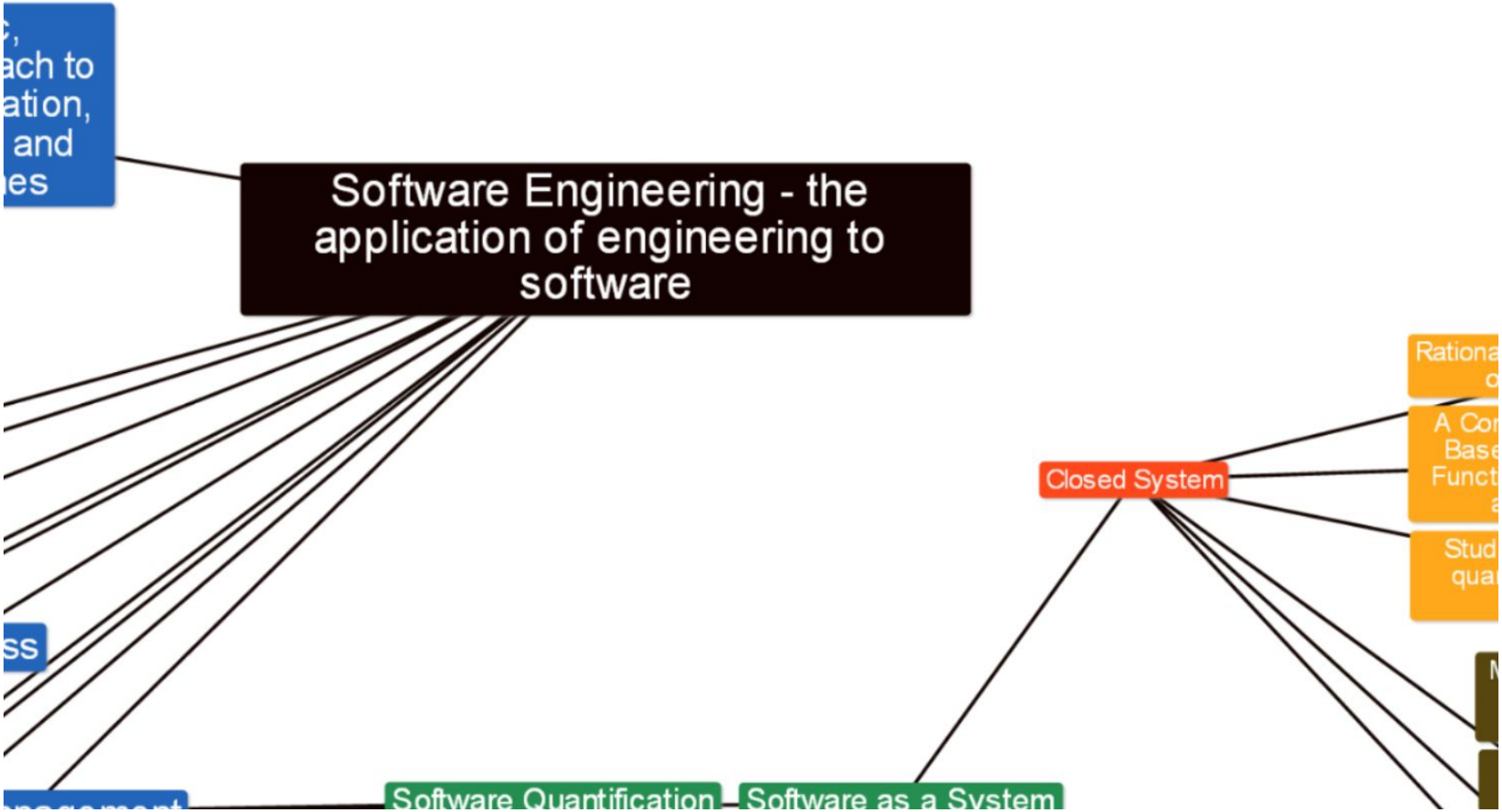
- High cohesion usually means low coupling – the degree to which each program module relies on each other
- It insures that the functional breakdown of the program reflects the functional organization of the original problem
- High cohesion has been shown to be a good predictor of maintainability

Correlations this study traces

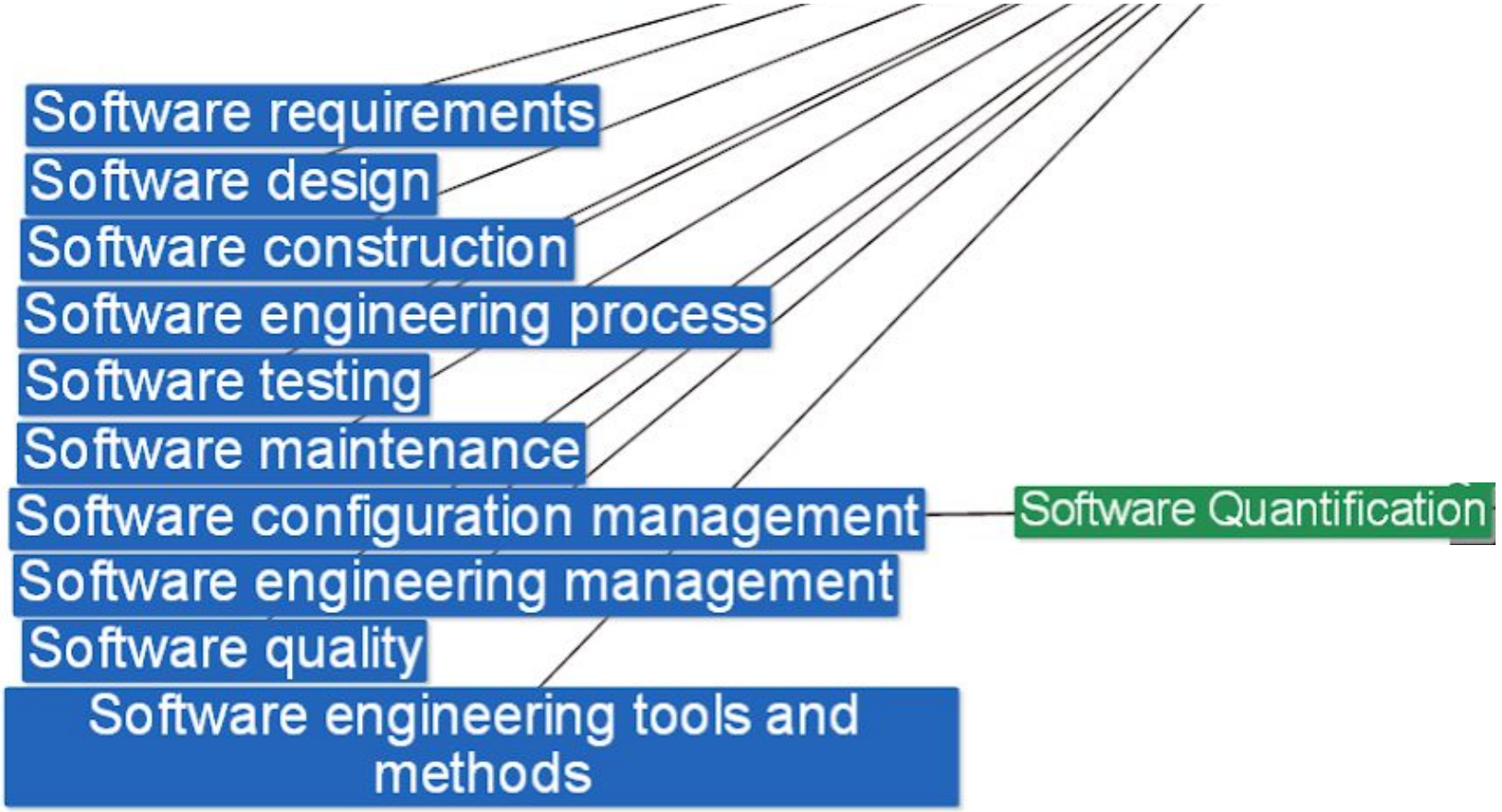


Sonia M. Pascua

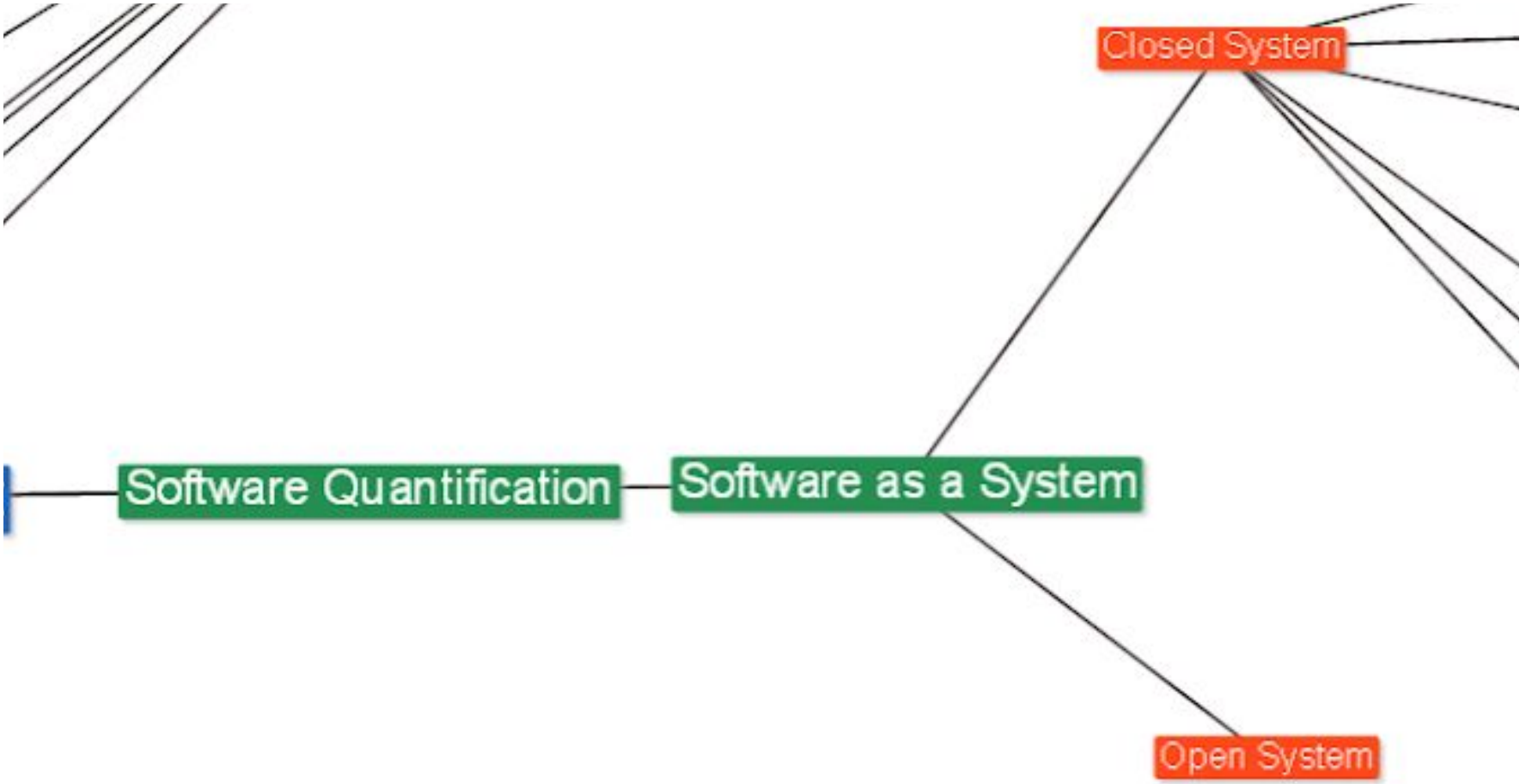
Correlations this study traces



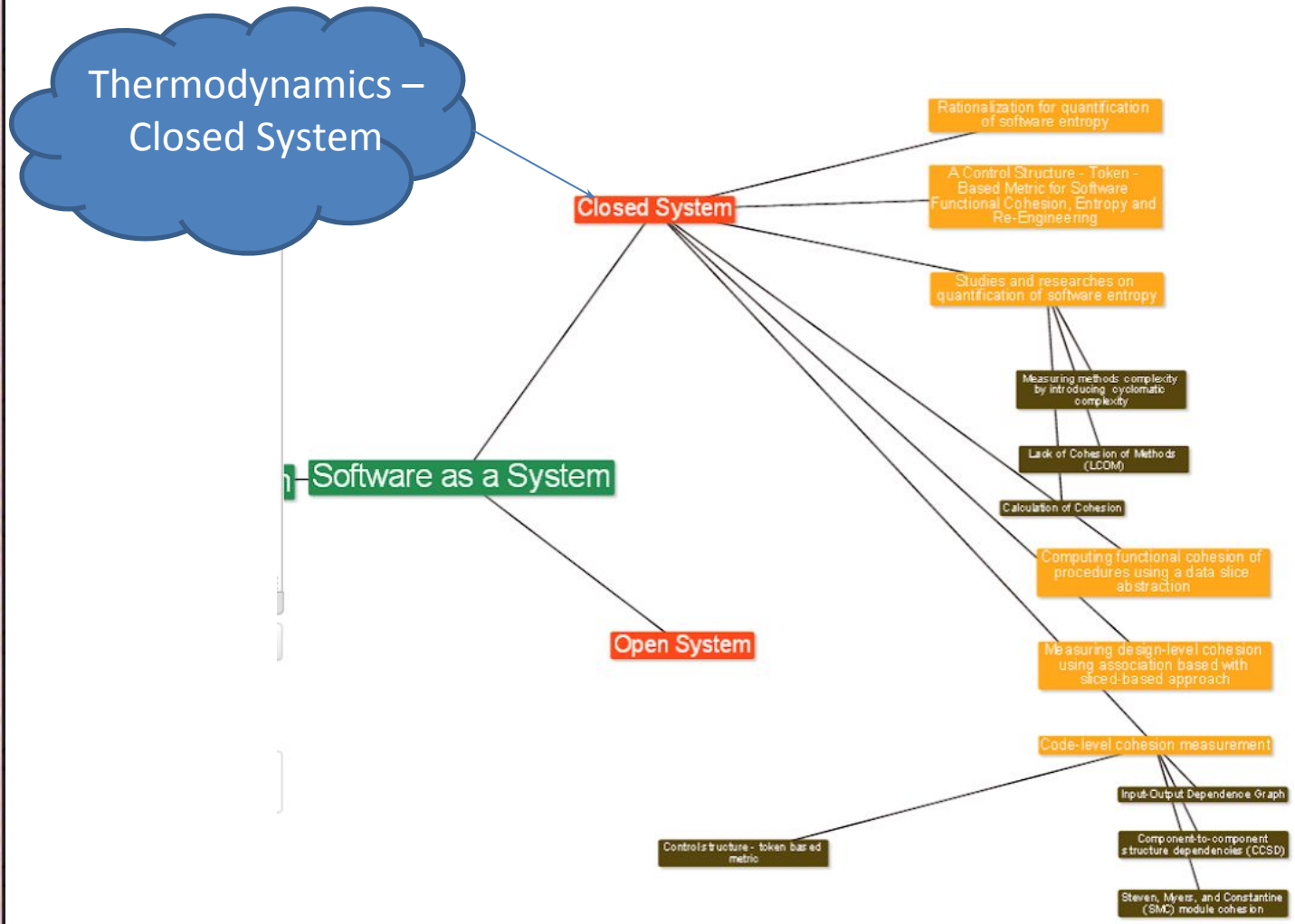
Correlations this study traces



Correlations this study traces



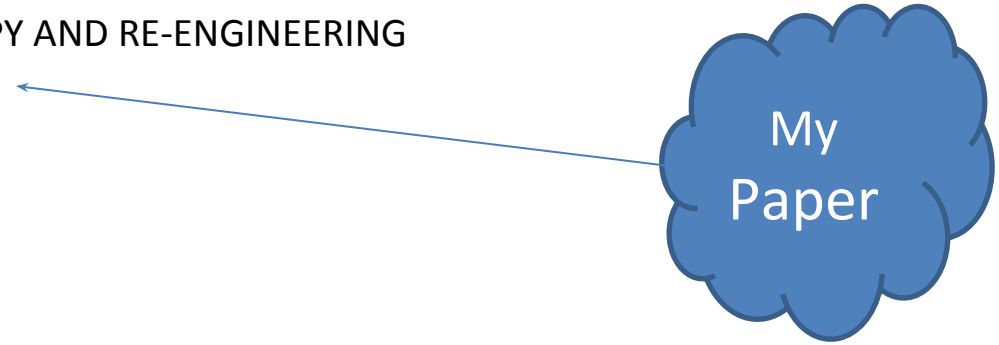
Correlations this study traces



Topics Covered

- Rationalization for quantification of software entropy
- Related studies and researches on quantification of software entropy
  - Chidamber and Kemerer 1991 , Quantifying Methods Complexity by introducing Cyclomatic Complexity
  - Chidamber and Kemerer 1994, Lack of Cohesion of Methods (LCOM)
  - Beiman and Ott 1994, examined functional cohesion of procedures using data slice abstraction
  - Derived measurements from these study is Yourdon and Constantine's
    - Strong Functional Cohesion (SFC)
    - Weak Functional Cohesion (WFC)
  - Beiman and Kang 1997, introduced Measuring Design-Level Cohesion by using association-based and sliced-based approaches
- **Code-level Cohesion Measurement**

  A CONTROL STRUCTURE-TOKEN-BASED METRIC FOR SOFTWARE FUNCTIONAL COHESION, ENTROPY AND RE-ENGINEERING



Enhancement of Existing Works

METHODOLOGY

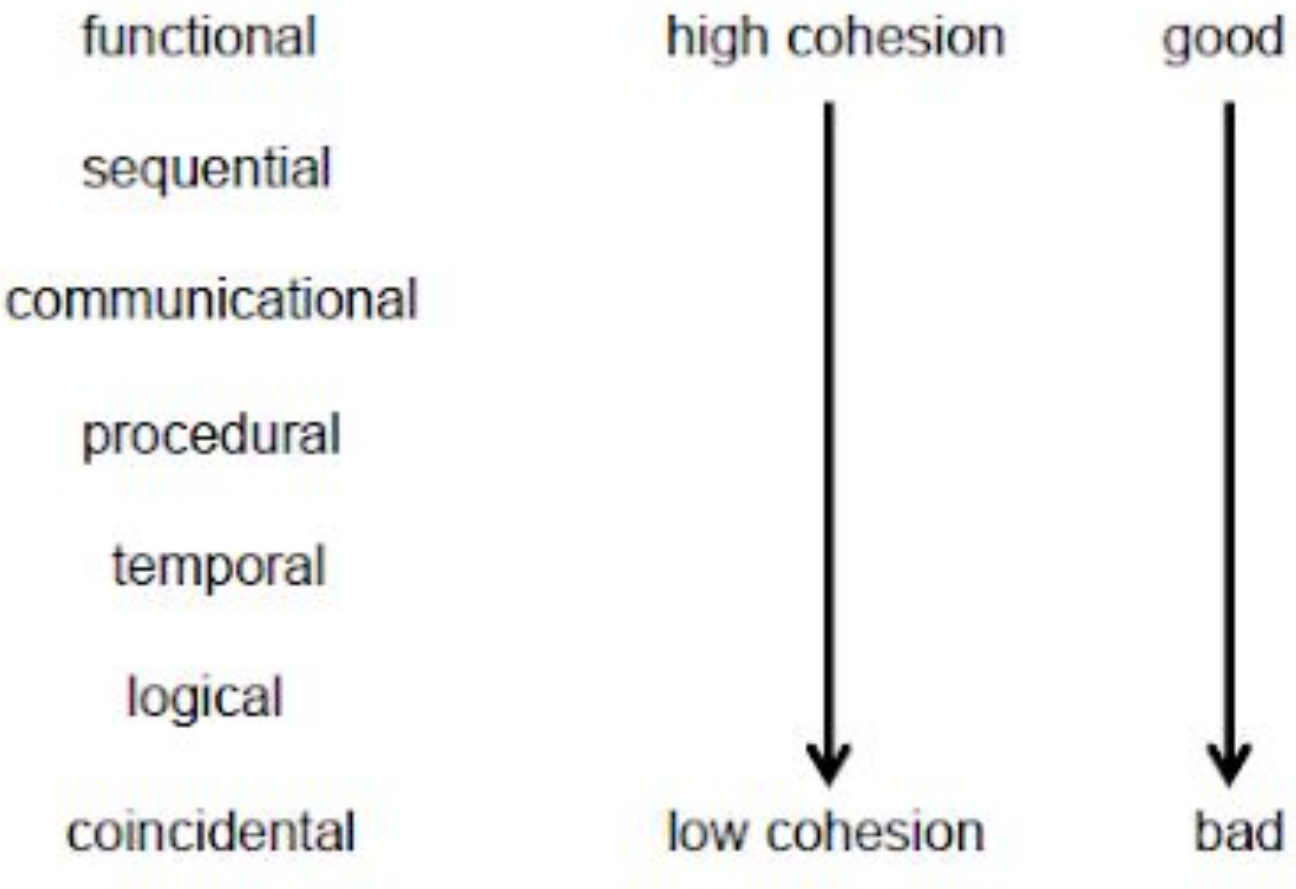
Related Works

- As a related study, the Measuring Design-Level Cohesion of Beiman and Kang showed that an analytical and empirical analysis on the design level measurement corresponded closely with the code-level cohesion measurement.
- Therefore Input Output Dependence Graph (IODG) which was adapted from Lakhotia's variable dependence graph defined the relationship between input and output components of a module causal to cohesion measurement.

Related Works

- Aside from deriving calculation for cohesion, Steven, Myers, and Constantine introduced module cohesion (SMC Cohesion) that is measurement of cohesiveness represented on an ordinal scale approach
- In this study modules are categorized among
  - Coincidental cohesion
  - Logical cohesion
  - Temporal cohesion
  - Procedural cohesion
  - Communicational cohesion
  - Sequential cohesion
  - Functional cohesion
- Thus stating coincidental cohesion regarded as the weakest and functional cohesion the strongest cohesion

SMC Ordinal Types of Cohesion



The Objective

- The contribution of this study, therefore, focuses on building a metric to enhance the real world implementation of sound engineering theories and practices by formulating and quantitatively measuring the concept known as Software Entropy by measuring control structures to variables cohesion.

The Objective

- This aims to help software designers, developers, and system managers to pinpoint when their update in LOCs causes entropy to the stability and robustness of their current software.

The Objective

- This could also help in rationalizing software update, partial or total re-engineering and /or migration.

The Methodology

- This study will run both the IODG methods and the SMC ordinal cohesion scale but
  - Unlike IODG that accounts Input and Output variables only
  - Unlike Data Slices approach that accounts all variables only
- This research will consider all **variables** and **control structures** in a module in computing cohesion

The Methodology

- Then the computed cohesion will be compared against the SMC ordinal cohesion scale to determine the **COHESION METRIC**

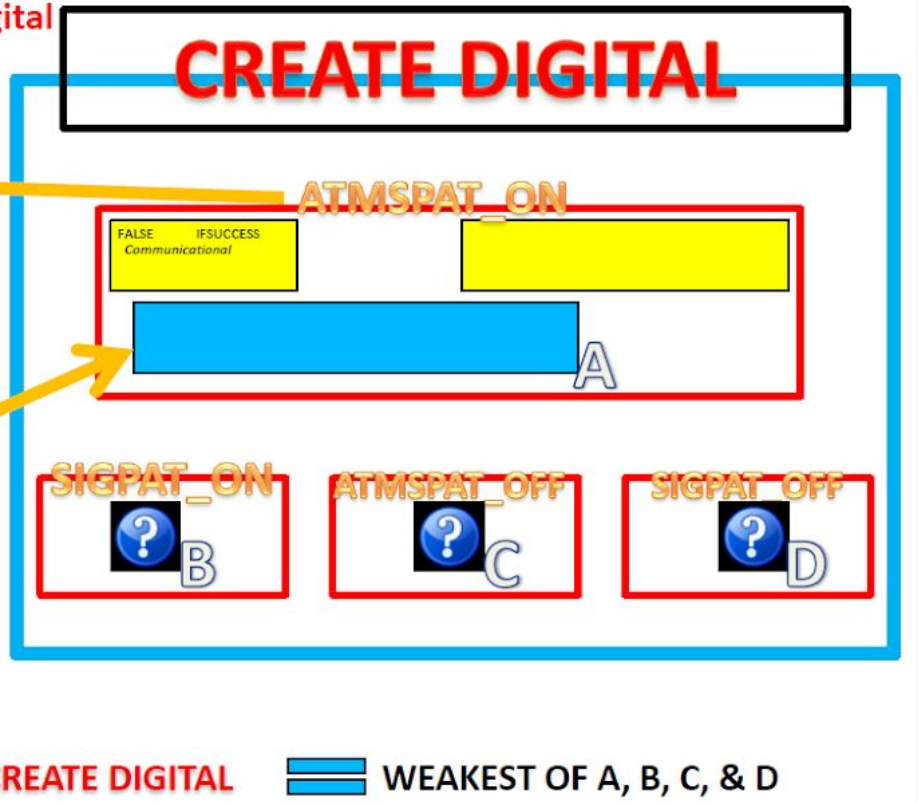
Visualization

Determining Cohesion Level of CREATE DIGITAL

1. Start with a Method of Create Digital
2. Get 1 pair of outputs and Determine the Strongest Cohesion using IODG

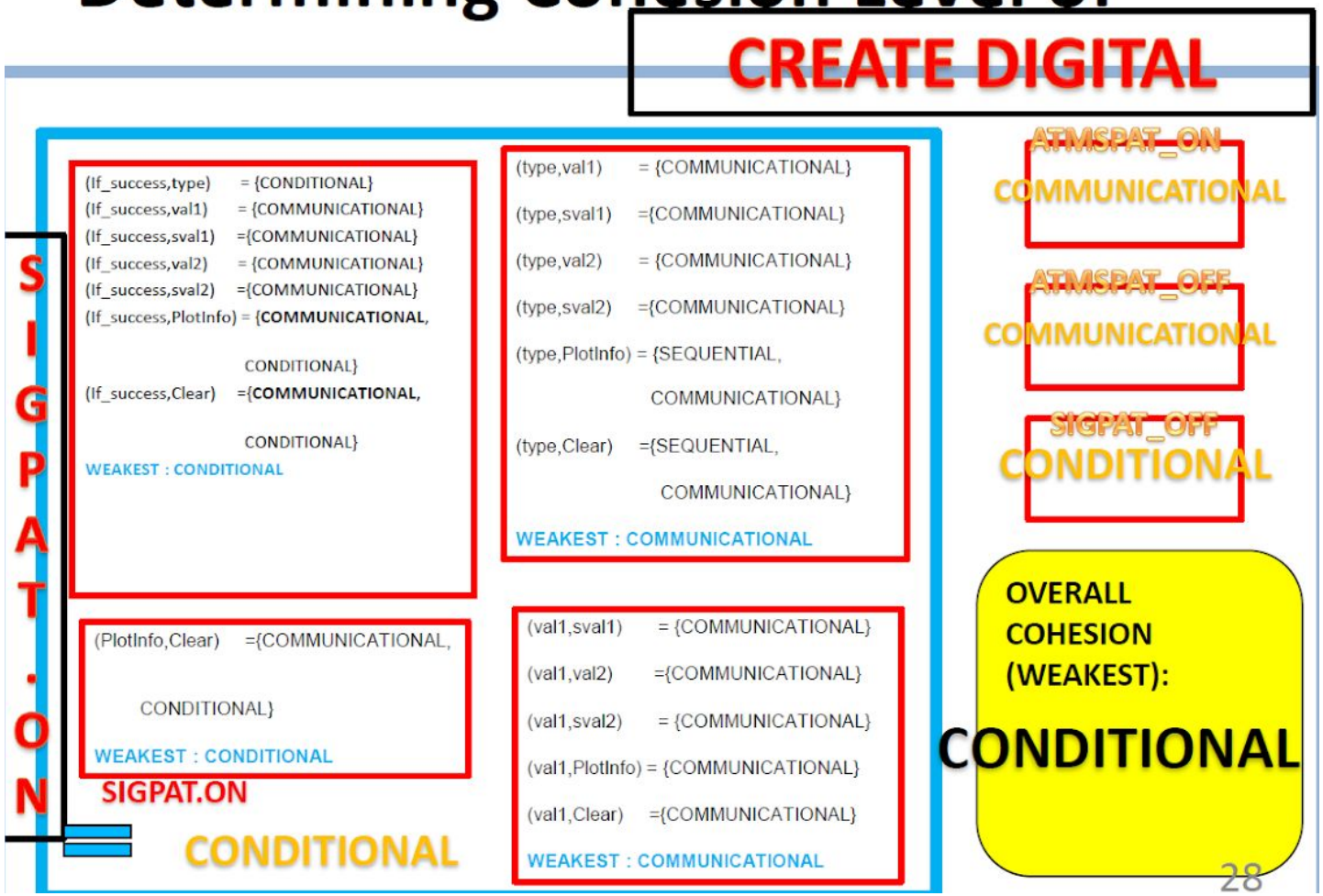
- ❑ *Coincidental Cohesion*
- ❑ *Conditional Cohesion*
- ❑ *Iterative Cohesion*
- ❑ *Communicational Cohesion*
- ❑ *Sequential Cohesion*
- ❑ *Functional Cohesion*

2. Repeat Step 2 for all other Pair of Outputs
3. Get the Weakest
4. Repeat Steps 1 to 3 for other Methods for Main Method: Create Digital
5. Get the Weakest



Visualization

Determining Cohesion Level of



CONTROL STRUCTURE – TOKEN DEPENDENCIES

Definition of Terms

- Token

Let component y be a variable or a constant,

$$\text{denoted by } y \in \{\text{variable, constant}\}$$

- Control Structure

Let control structure c be either if, while or do-while,

$$\text{denoted by } c \in \{\text{if, while, do-while}\}$$

NINE (9) DATA DEFINITIONS

ALGORITHMS

Data Definition

(1) A component y is *data dependent* on a control structure c,

denoted by $y \xrightarrow{d} c$,

if the modification of the value/size/type of y is dependent on the satisfiability of c.

(2) A control structure c has an *execution control dependence* on component y,

denoted by $c \xrightarrow{ec} y$,

if y's value influences the execution of c's body.

Data Definition

(3) A component y is dependent on a control structure c (or vice versa),

denoted by $y \longrightarrow c$ or $c \longrightarrow y$,

when there is a path from c to y (or from y to c) through a sequence of data or execution control dependence. We call such path a *dependence path*.

(4) A component y has *i-dependence* on a control structure c,

denoted by $y \xrightarrow{id} c$,

if c has an execution control dependence on y, $c \xrightarrow{ec} y$ and c is iterated.

Data Definition

(5) A component y has *p-data dependence* on a control structure c,

denoted by $y \xrightarrow{pd} c$,

if y is data dependent on c , $y \xrightarrow{d} c$ and there is a dependence path from c to y, $c \longrightarrow y$ or dependence path from y to c, $y \longrightarrow c$.

(6) A control structure c has an *iteration execution control dependence* on y,

denoted by $c \xrightarrow{iec} y$,

if c has an execution control dependence on y, $c \xrightarrow{ec} y$ and y's value influences the iteration of c.

Data Definition

(7) A control structure c has a *c-path dependence* on y,

denoted by $c \xrightarrow{cp} y$,

if there is a dependence path from y to c, $y \longrightarrow c$, such that y has an execution control dependence on c.

(8) A component y has *ip-data dependence* on a control structure c,

denoted by $c \xrightarrow{ip-d} y$,

if y has p-data dependence on c, $y \xrightarrow{pd} c$ and c is iterated.

Data Definition

(9) A component y has ic-path dependence on c,

denoted by $c \xrightarrow{i-cp} y$,

if c has a c-path dependence on y, $c \xrightarrow{ip} y$ and c is iterated.

CSTDG

CONTROL STRUCTURE – TOKEN DEPENDENCE GRAPH

CSTDG

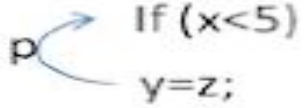
The *control structure to component dependence graph* of a module M is a directed graph, denoted by $G_M = (V, E)$ where V is set of components of M, and E is a set of edges labeled with dependence types such that

$E = (y, c) \in V \times V \mid$ y has data, c is any control structure;

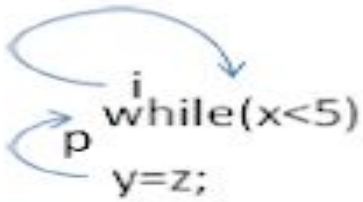
control structure c can be either of the following:

Control Structure

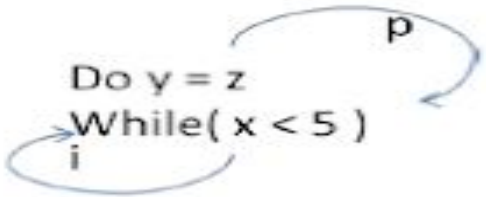
(1) *If Control Structure*



(2) *While-loop Control Structure*



(3) *Do-while loop Control Structure*



EMPIRICAL DATA

COMPUTING COHESION

Data Slice Profile for IF

Statement	Output z
procedure IF	
(x : integer;	1
y : integer;	1
z : integer;	1
begin	
x:= 0 to N	2
y:= 0	2
z:= 0 do begin	2
$if(x < N)$	3
y:= z	3
end;	
end;)	

Data Slice Profile for IF

Statement	Output z
procedure IF	
(x : integer;	1
y : integer;	1
z : integer;	1
begin	
x:= 0 to N	2
y:= 0	2
z:= 0 do begin	2
$if(x < N)$	3
y:= z	3
end;	
end;)	

- Functional Cohesion is measured based from the procedure output and the components of the procedure that contribute to the said output.

Data Slice Profile for IF

Statement	Output z
procedure IF	
(x : integer;	1
y : integer;	1
z : integer;	1
begin	
x:= 0 to N	2
y:= 0	2
z:= 0 do begin	2
$if(x < N)$	3
y:= z	3
end;	
end;)	

- Using this approach, it is ensured that maximum functional cohesiveness is achieved.
- For the cases of multiple outputs in a procedure, each components contributing to different outputs will be considered together with how they are bound.

Results and Findings

- To apply the *Data Slice Abstraction*, the above table shows several data tokens contributing to a data slice, which is the *Output z*

- and where data tokens are the components or the statements in the above tables.

Cohesion Computation

Data slice for Output z: $x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3$

Slice Abstraction (SA) for IF Control Structure = ($x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3$)

Since the distribution of *glue* and *supper-glue* tokens indicates how tightly bound an individual slices are. They are used to develop measurement for the computation of functional cohesion for the IF Control Structure.

The data slice profile above shows that the IF Control Structure has only one slice. This indicates results below:

Glue token in a SA of IF procedure $\mathbf{P,G(SA(P))}$ = Super Glue token in SA of IF procedure $\mathbf{SG(SA(P))}$

And therefore SG(SA(P)) $\geq$ P,G(SA(P)) means all super-glue tokens are all glue tokens.

Table below shows Glueness of the data token for IF procedure

Data Token	Glueness
$procedureIF$	
(x : integer;	|
y : integer;	|
z : integer;	|
begin	
x:= 0	|
y:= 0	|
z:= 0	|
$if(x < 5)$ do begin	|
y:= z	|
end;	
end;)	

Data Slice Profile for WHILE

Statement	y	i
procedure WHILE		
(x : integer;	1	
y : integer;	1	1
i : integer;	1	1
begin		
x:= 0 to i	2	
y:= 0	2	2
i:= 0 do begin	2	2
$while(x < i)$	3	3
{y:= i	3	3
i++}	3	3
end;		
end;)		

Data Slice Profile for DO WHILE

Statement	y	i
procedure DO-WHILE		
(x : integer;	1	
y : integer;	1	1
i : integer;	1	1
begin		
x:= 0 to i	2	
y:= 0	2	2
i:= 0 do begin	2	2
Do (y:= i)	3	3
{*while*$(x < i)$	3	3
i++}	3	3
end;		
end;)		

Conclusions

- Clearly, by incorporating these 9 types of relationships among control structures and tokens, we augment the information we can derive regarding the cohesiveness of the elements within a module m.
- However, knowing when and how to refactor codes is an issue of the skill of the designer and/or programmer to discern.
- This paper would measure cohesion within a procedure and compare it to the SMC Cohesion ordinal scale.
- However entire code should undergo repetitive computation to really see the exact SMC Cohesion level.

Future Works

- Automation of the computation of the functional cohesion
- An Assembly-Level Procedural Cohesion (Hardware)

Acknowledgements

This research work is a result of the partnership of the University of the Philippines Information Technology Development Center (UP ITDC) and Fujitsu Ten Limited represented by Fujitsu Ten Solutions Philippines, Inc.

References

(1) Rosenberg, Doug and Stephens, Matt. Use Case Driven Object Modeling with UML: Theory and Practice. Berkley: Apress, 2007.

(2) S. R. Chidamber and C. F. Kemerer, (1991) Towards A Metrics Suite for Object-Oriented Design, *OOPSLA '91 Conference Proceedings , Special Issue of SIGPLAN Notices,* Vol. 26, No. 11, November 1991, pp. 127 - 211.

(3) S. R. Chidamber and C. F. Kemerer, (1994) A Metrics Suite for Object-Oriented Design, *IEEE Transactions on Software Engineering,* Vol. 20, No. 6, June 1994, pp. 476 - 493.

(4) J. Beiman and L. Ott, "Measuring Functional Cohesion", Tech. Rep. CS-93-109, Comput. Scie. Dept., Colorado State Univ. Tech. REp. CS-93-1, Comput. Sci. Dept., Michigan Technological Univ.,1993.

(5) E. Yourdon and L. Constantine, Structured Design. *Englewood Cliffs,* NJ: Prentice Hall, 1979.

(6) J. Beiman and B. K. Kang, "Measuring Design-Level Cohesion", Comput. Scie. Dept., Colorado State Univ., Elec. and Telcomm. Res. Inst., 161 Kajong-Dong, Yusong-Gu, Taejon, 305-350 Korea.,1997.

(7) A. Lakhotia, "Rule-Based Approach to Computing Module Cohesion", *Proc. 15th Int'l Conf. Software Eng.,* pp. 35-44, 1993.

(8) W. Stevens, G. Myers, and L. Constantine, "Structured Design", *IBM Systems J.,* vol. 13, no. 2, pp. 115-139, 1974.

(9) N. Gupta," Automated Software Engineering, 2001",*ASE 2001 Proceedings.* 16th Annual International Conference.

(10) R. Sobiesiak and Y. Diao, "Quantifying Software Usability Through Complexity Analysis, 2010", *IBM Corporation.*

(11) Singh, Pariksha, "Validating Cohesion Metrics by Mining Open Source Software Data With Association Rules", *Durban University.*


THANK YOU! MARAMING SALAMAT PO!