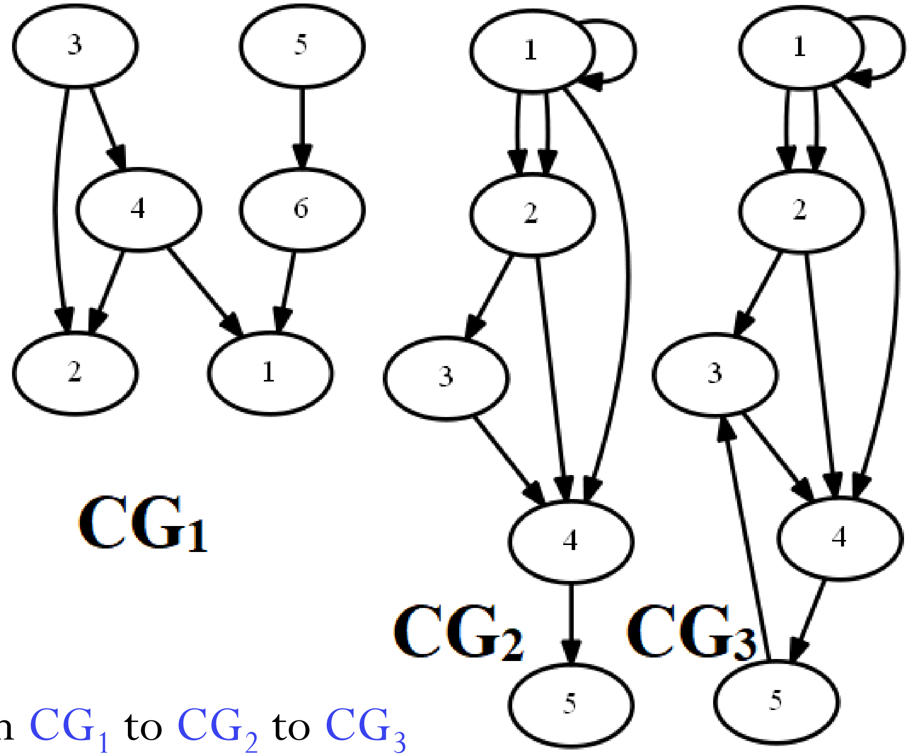
37th IEEE/ACM International Conference on Automated Software Engineering
ASE 2022

Call Graph Evolution Analytics over a Version Series of an Evolving Software System

Dr. Animesh Chaturvedi
Assistant Professor: IIIT Dharwad
Young Researcher: Heidelberg Laureate Forum
Postdoc: King's College London & The Alan Turing Institute
PhD: IIT Indore MTech: IIITDM Jabalpur

Abstract view

- ***Call Graph Evolution Analytics***
  - extracts information from a set of *Evolving Call Graphs* ECG = $\{CG_1, CG_2, \ldots CG_N\}$
  - ECG represents a Version Series VS = $\{V_1, V_2, \ldots V_N\}$ of an evolving software system.
- ***Call Graph Evolution Rules* (CGERs)**
  - similar to association rule mining,
  - the CGERs are used to capture co-occurrences of dependencies in the system.
- ***Call Graph Evolution Subgraphs* (CGESs)**
  - like subgraph patterns in a call graph,
  - the CGESs are used to capture evolution of dependency patterns in evolving call graphs.



Evolution of a call graph from CG_1 to CG_2 to CG_3
when a software evolves from version V_1 to V_2 to V_3.

Software Repository Evolution

- Maintenance phase, a software repository holds plenty of information about the dependency evolution in call graphs.
- This study focuses on dependency repositories (e.g. Maven and GitHub), which contain jars, APIs, and libraries.
- Growth of software
  - in the numbers of repositories and
  - in the size of repositories
  - makes mining of a dependency repository a challenge.
- Existing techniques commonly analyse a single software version.

Call Graph

- A call graph represents procedure calls (or dependency) relationships in the form of a directed graph, where
  - nodes represent procedures (method or function) and
  - directed edges represent dependencies from caller to callee procedures.
- Earlier call graph analysis techniques have proven to be helpful in many software engineering applications.
- The evolution of procedure calls can be exploited using techniques such as change mining and evolution mining.
- Call graph analytics on the evolution in procedure patterns can identify potentially affected dependencies (or procedure calls) that need attention.

Call Graph Evolution Analytics

Call Graph Evolution Analytics

- **Call Graph Evolution mining** techniques to uncover interesting and useful information.
- On a **fine-grained level**,
  - the **rule mining** retrieves co-occurrences of dependencies in a call graph of a version, and then counts stability of retrieved rules over the version series.
- On a **coarse-grained level**,
  - the **subgraph mining** extracts frequently occurring structural patterns of dependencies in a call graph of a version, and then aggregates frequencies of retrieved patterns over the version series.

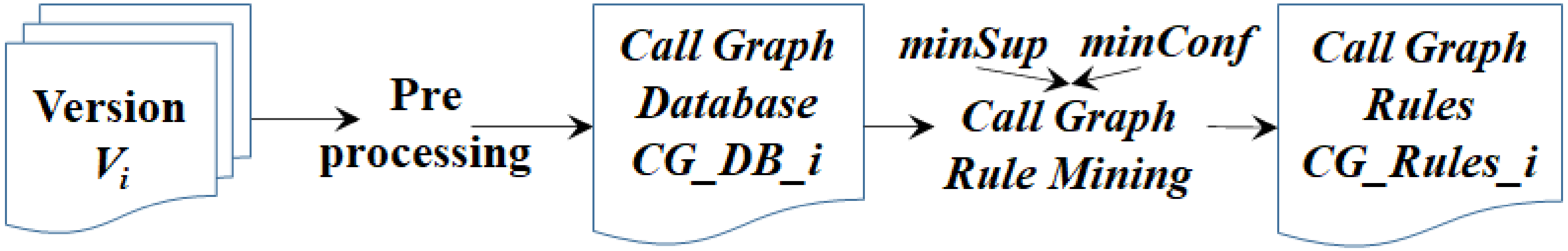
Call Graph Evolution Analytics

- **Software evolution analytics**
  - can be supported by generating and comparing call graph evolution information over versions of a software system.
  - apply graph evolution mining on multiple call graphs representing multiple versions of a software repository.
- **Call graph evolution analytics**
  - provides information about the software evolution.
  - information enables tools to support and manage the evolution of dependencies.
  - can assist a software engineer when maintaining or evolving a software system.
- **Research Question**
  - "How does the results correspond to the **Lehman's law of software evolution**?"

Call Graph Evolution Rules (CGERs)

Call Graph Database

- Given a version of a system,
  - the procedures of the system are assumed to be divided up into modules
  - "procedures-module membership information"
  - transform this information into a set of "call pairs",
  - form the Call Graph Database CG_Db.
  - use this database with two rule mining thresholds,
    - minimum support (minSup) and
    - minimum confidence (minConf)
  - generate a collection of Call Graph Rules (CG_Rules_i)



Call Graph Rule

- The Call Graph Rule X→Y can be interpreted as
  - "if the procedure(s) of set X appear in a call pair, then the procedure(s) of set Y are likely to appear in the module with support and confidence above minSup and minConf, respectively".
  - the presence of the caller procedure(s) in X implies the presence of the callee procedure(s) in Y with sufficiently high frequency to surpass the two thresholds minSup and minConf.
- The CGR is a rule
  - with its support and confidence.
  - interesting in a version V_i.
  - derived from a call graph of a software version.

Call Graph Evolution Rules (CGERs)

- Select distinct CGRs as CGERs
  - form a collection of distinct CGRs in multiple versions
- Count of a reoccurring CGR in multiple versions depict its
  - stability over a version series.
- The CGERs accompany the count (as stability).
- Each CGER has its 'stability',
  - the count of versions in which CGR appears interesting (i.e., its support and confidence is above minSup and minConf).
- Earlier, we defined minimum Stability (**minStab**) [34][35].

[34] A. Chaturvedi, A. Tiwari, and N. Spyratos. "minStab: Stable Network Evolution Rule Mining for System Changeability Analysis". *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.2 (2019): 274-283.
[35] A. Chaturvedi, A. Tiwari, and N. Spyratos. "System Network Analytics: Evolution and Stable Rules of a State Series". *IEEE 9th International Conference on Data Science and Advanced Analytics* (DSAA). IEEE, 2022.

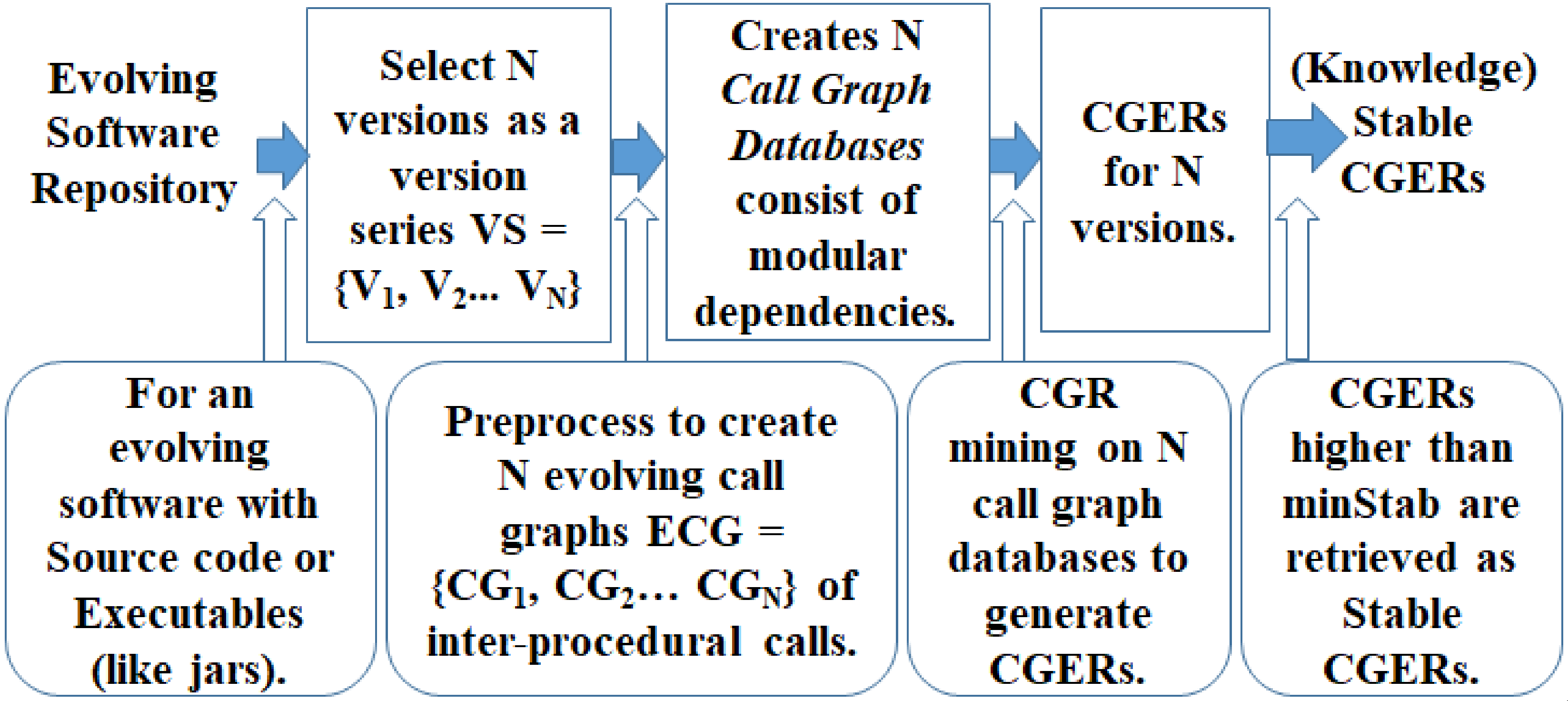
Stable CGERs

- A CGER is defined as a Stable CGER in VS, if

  (a) the support and confidence are greater than minSup and minConf in a version V_i, and

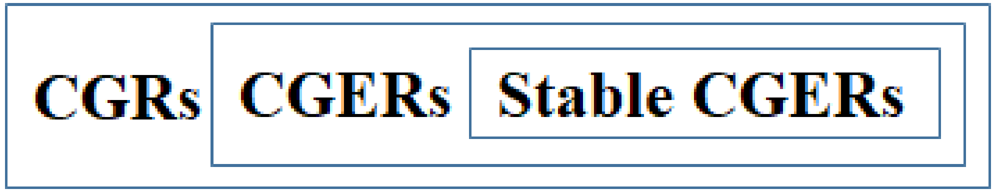
  (b) the 'stability' > minStab (i.e. stability of CGERs is greater than the minStab).

- A CGER is stable in VS if it exceeds the three user-specified thresholds:
  - minSup,
  - minConf, and
  - minStab



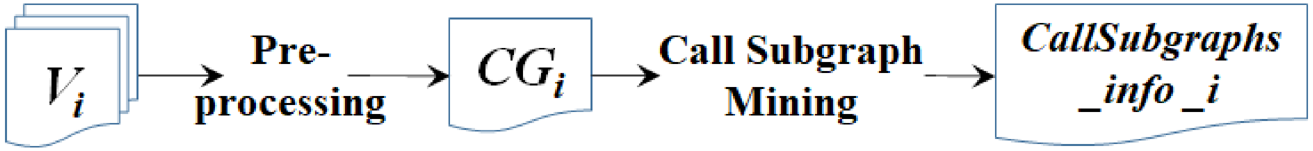
Stable CGERs Mining over a version series

- These CGERs can also be referred to as
  - call graph prediction rules (temporal rules, or episode rules)
- This takes the general form
  - "if a certain dependency(ies) occurs, then another dependency(ies) is(are) likely to occur in the module".
- There will be a subset relationship between
  - CGRs,
  - CGERs, and
  - SCGERs.



Call Graph Evolution Subgraphs (CGES)

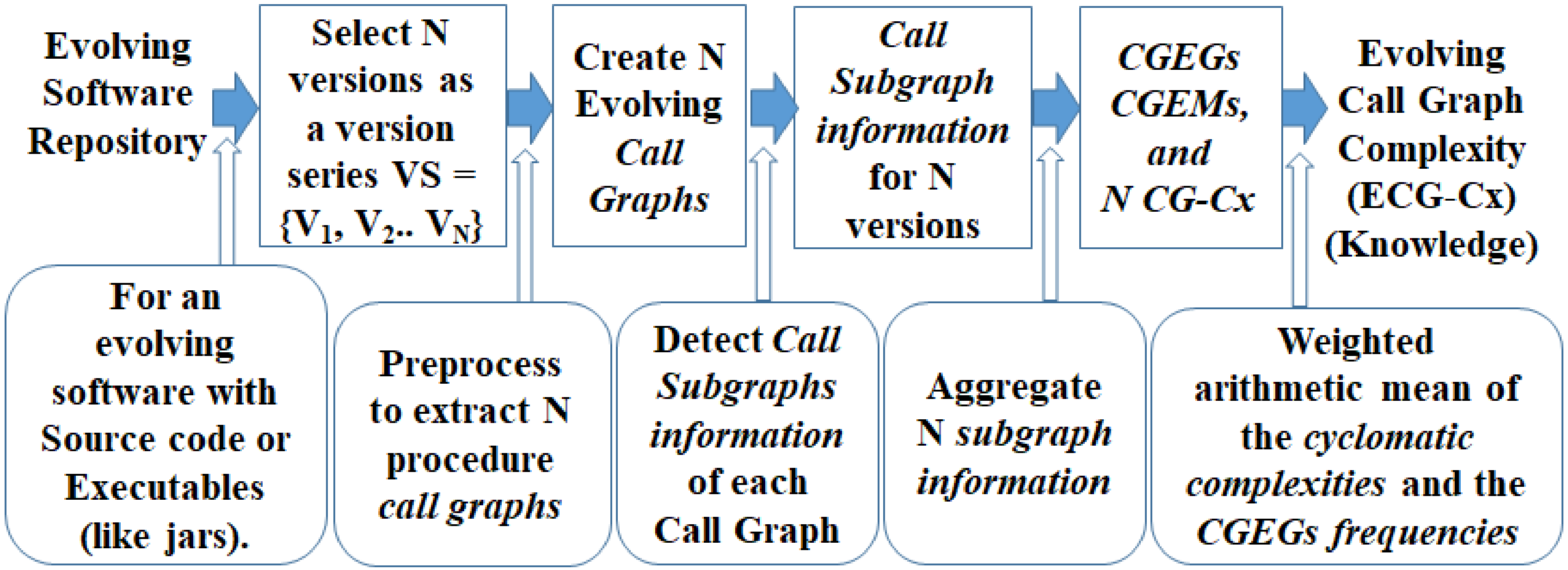
Call Graph Evolution Graphlets



- Software version V_i
  - pre-process to build its call graph
  - a call graph is the input for subgraph mining
    - retrieves subgraph patterns with their frequencies
  - apply subgraph mining to each of the call graph in a set of N call graphs
    - retrieve the graphlets information for all versions of a software to extract the Call Graph Evolution Subgraphs and their frequencies.
- These subgraphs are of two types
  - **Call Graph Evolution Graphlets (CGEGs)**, and
  - **Call Graph Evolution Motifs (CGEMs)**

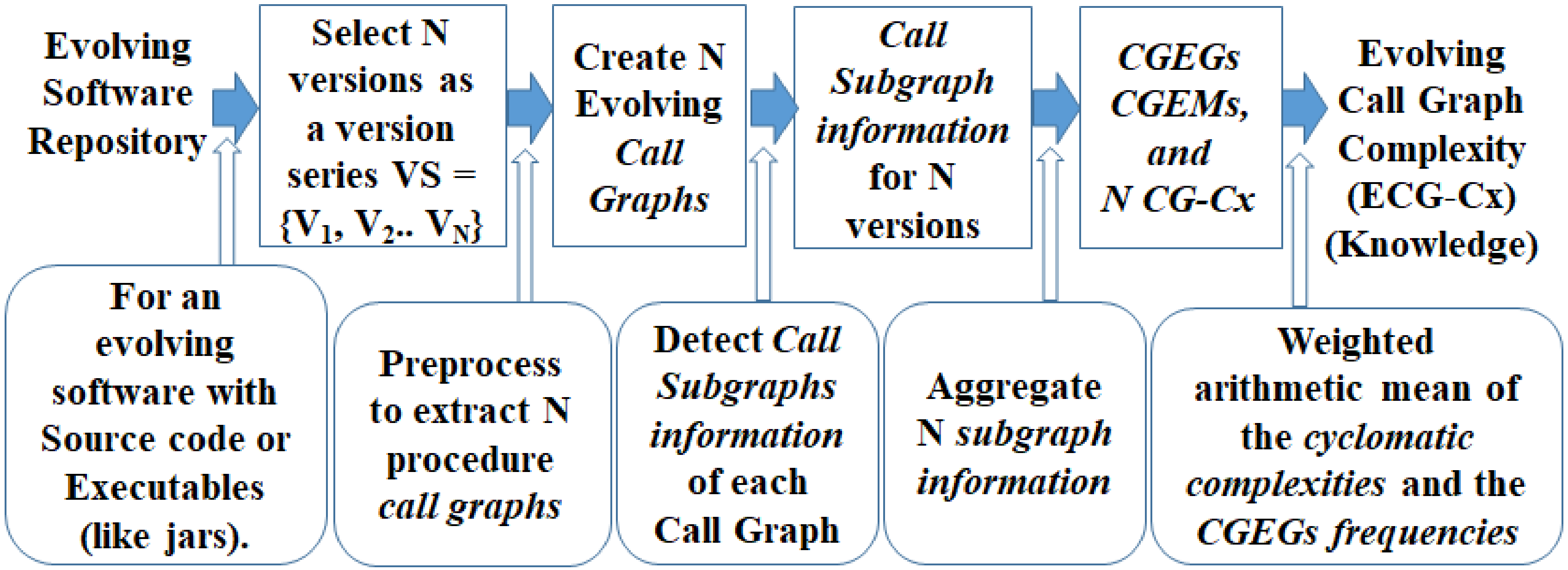
Complexity analysis

- The Call Graph Evolution Graphlet information (subgraphs and their frequencies) are used to calculate
  - a Call Graph Complexity (**CG-Cx**) of a single version and
  - the overall Evolving Call Graph Complexity (**ECG-Cx**) for a version series.
- Knowledge Discovery and Data-mining (KDD) steps to detect
  - CGEGs,
  - CGEMs,
  - CG-Cx,
  - ECG-Cx



Complexity analysis

- Discovers hidden dependency evolution patterns, which is captured in
  - CGEGs
  - CGEMs
- The frequencies of graphlets are meaningful quantities that are used to calculate complexity for a call graph.



Evolving Call Graphs Analytics

Information about Evolving Call Graphs

- 10 large evolving software
  - version series used to perform experiments,
  - number of procedures in those versions,
  - average number of neighbors of each procedure

Evolving software	Evolving Call Graphs		
	Version series	# procedures	Average # neighbours
Commons Codec	{1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10}	162	1.995
Guava	{12, 13, 14, 15, 16, 17, 18, 19}	1281	2.471
Hadoop HDFS	{2.2.0, 2.3.0, 2.4.0, 2.4.1, 2.5.0, 2.5.1, 2.5.2, 2.6.0, 2.6.1, 2.6.2, 2.6.3, 2.6.4, 2.7.0, 2.7.1, 2.7.2}	3129	2.166
HTTP Client	{4.3.1, 4.3.2, 4.3.3, 4.3.4, 4.3.5, 4.3.6, 4.3.0, 4.4.1, 4.4.0, 4.5.1, 4.5.2, 4.5.0}	276	2.020

Evolving software	Evolving Call Graphs		
	Version series	# procedures	Average # neighbours
JMeter Core	{1.8.1, 1.9.1, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13 }	806	2.632
Joda Time	{2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8.0, 2.8.1, 2.8.2, 2.9.0, 2.9.1, 2.9.2, 2.9.3}	418	2.886
JUnit	{4.1, 4.6, 4.7, 4.8, 4.9, 4.11, 4.12 }	171	1.628
Log4J	{1.1.3, 1.2.4, 1.2.5, 1.2.6, 1.2.7, 1.2.8, 1.2.9, 1.2.11, 1.2.12, 1.2.13, 1.2.14, 1.2.15, 1.2.16, 1.2.17}	307	2.320
Maven Core	{3.1.0, 3.1.1, 3.2.1, 3.2.2, 3.2.3, 3.2.5, 3.3.1, 3.3.3, 3.3.9}	594	2.385
Storm Core	{0.9.1, 0.9.2, 0.9.3, 0.9.4, 0.9.5, 0.9.6, 0.10.0, 0.10.1}	373	1.686

Number of

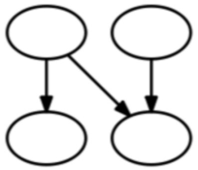
Evolving Call Graphs Analytics

- For each evolving software, make a series of evolving call graphs
- Apply the call graph evolution analytics based on the two techniques:
  - Stable CGERs mining
  - CGESs mining.
- Retrieve
  - CGRs, CGERs, Stable CGERs,
  - CGEGs, and CGEMs
  - persistence and complexity of dependencies in evolving call graphs.
- Out of 10 evolving software systems, detailed demonstration of **Maven-Core** is presented.

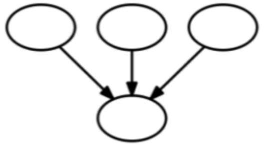
Experimental results on Maven-Core

- The 16 Stable CGERs retrieved for the Maven-Core.
- Out of many CGEGs, few statistically significant CGEMs patterns with their frequencies above a certain threshold.

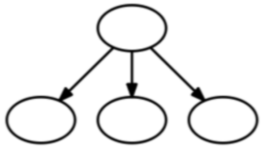
CGEMs



$(M_{198}, 39.55)$



$(M_0, 34.25)$

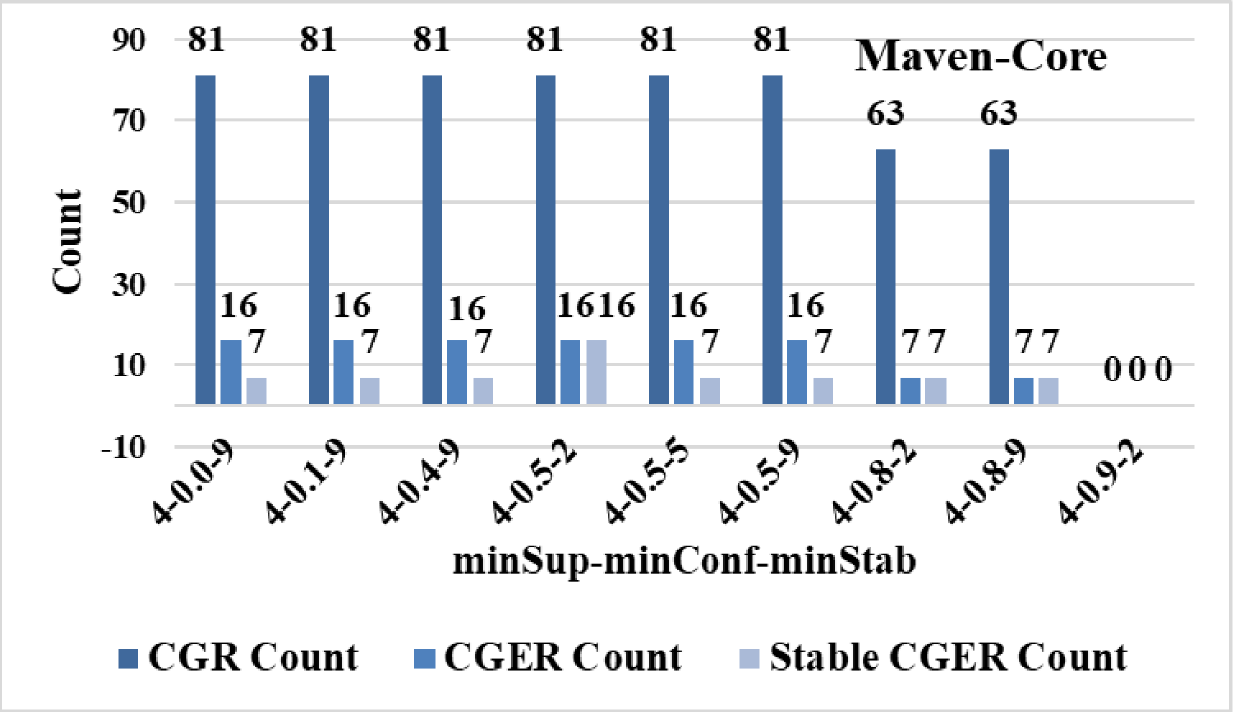


$(M_{178}, 13.85)$

Stable CGERs
{getGroupId → getArtifactId}
{getGroupId → getArtifactId, getId}
{getArtifactId → getGroupId}
{getArtifactId → getGroupId, getId}
{getGroupId → getId}
{getId → getGroupId}
{getId → getGroupId, getArtifactId}
{getKey → getGroupId}
{getKey → getGroupId, getId}
{getKey → getGroupId, getArtifactId}
{getKey → getGroupId, getArtifactId, getId}
{getArtifactId → getId}
{getId → getArtifactId}
{getKey → getArtifactId}
{getKey → getArtifactId, getId}
{getKey → getId}

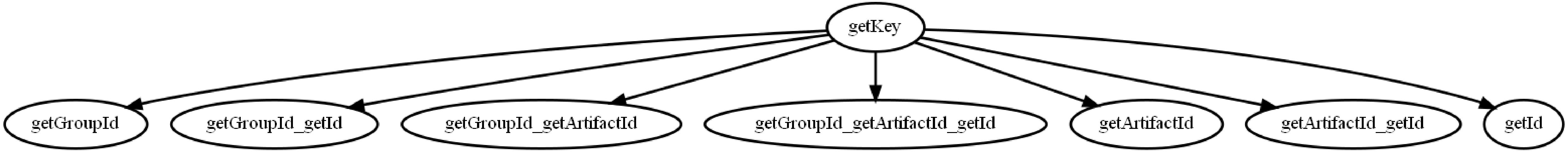
Compare CGRs, CGERs, & Stable CGERs.

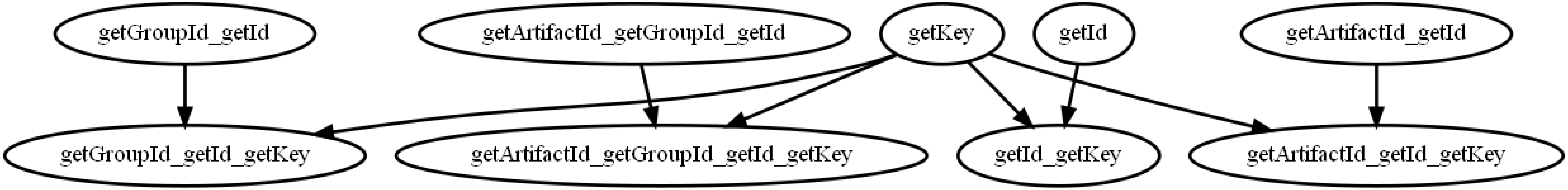
- Compare the number of CGRs, CGERs, and Stable CGERs for Maven-Core.
- The nine experiments
- Each experiment identified
  - interesting CGERs and Stable CGERs
  - Stable CGERs are fewer than CGRs



Transitivity and Lattice graphs

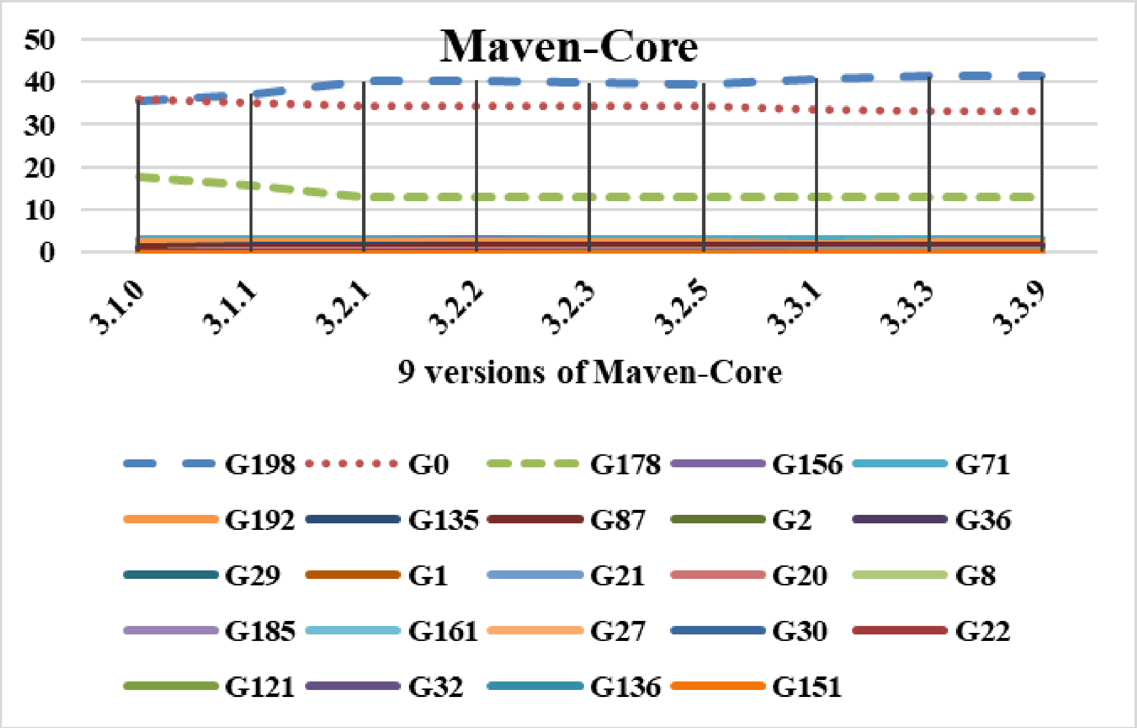
- For given Stable CGERs for the Maven-Core (evolving software).
- Some of the transitivities and lattices formed based on these Stable CGERs,
- Relationships between the various procedures in the Maven-Core.





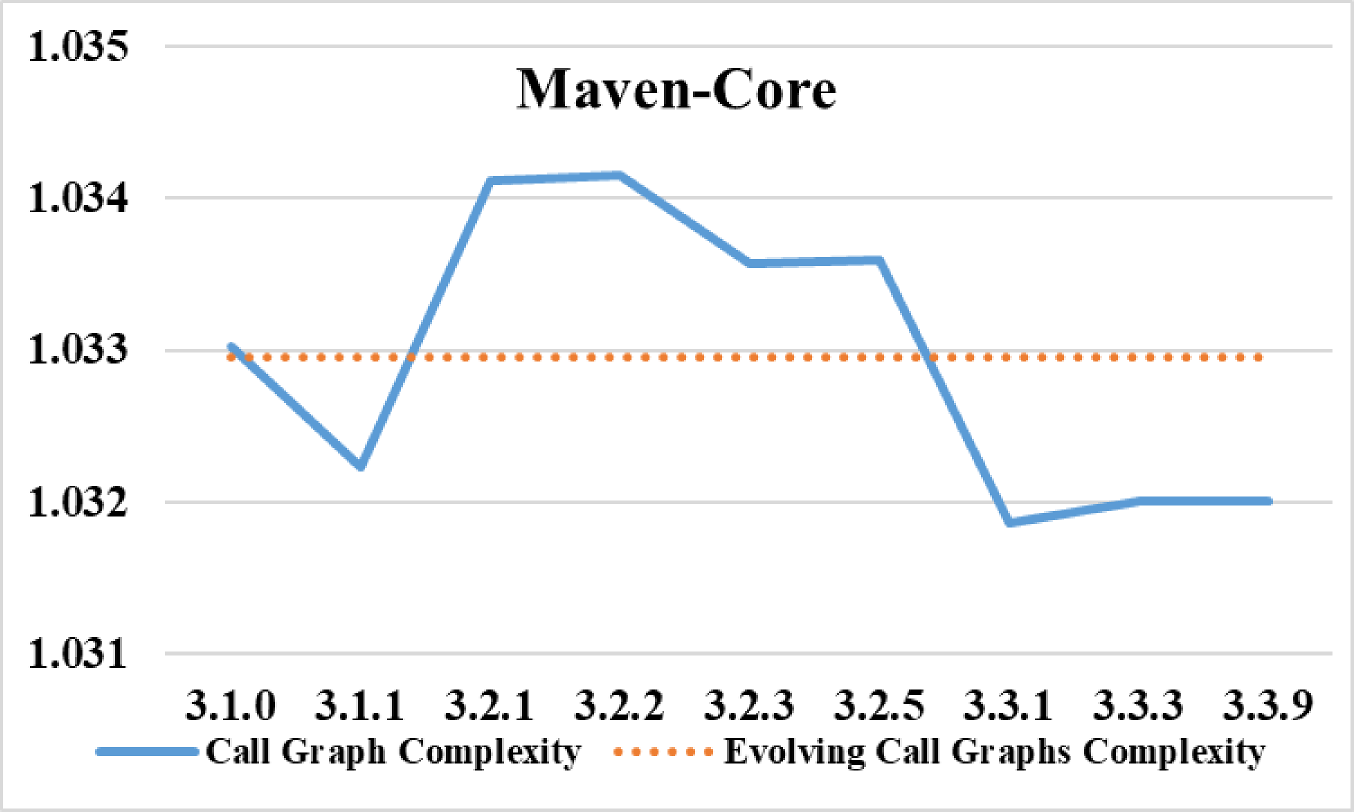
Frequencies of CGEGs

- Frequencies of various CGEGs of
  - evolving call graphs representing the version series of Maven-core.
- Various CGEGs are retrieved from the evolving call graphs
- The CGEGs with their frequencies over a version series.



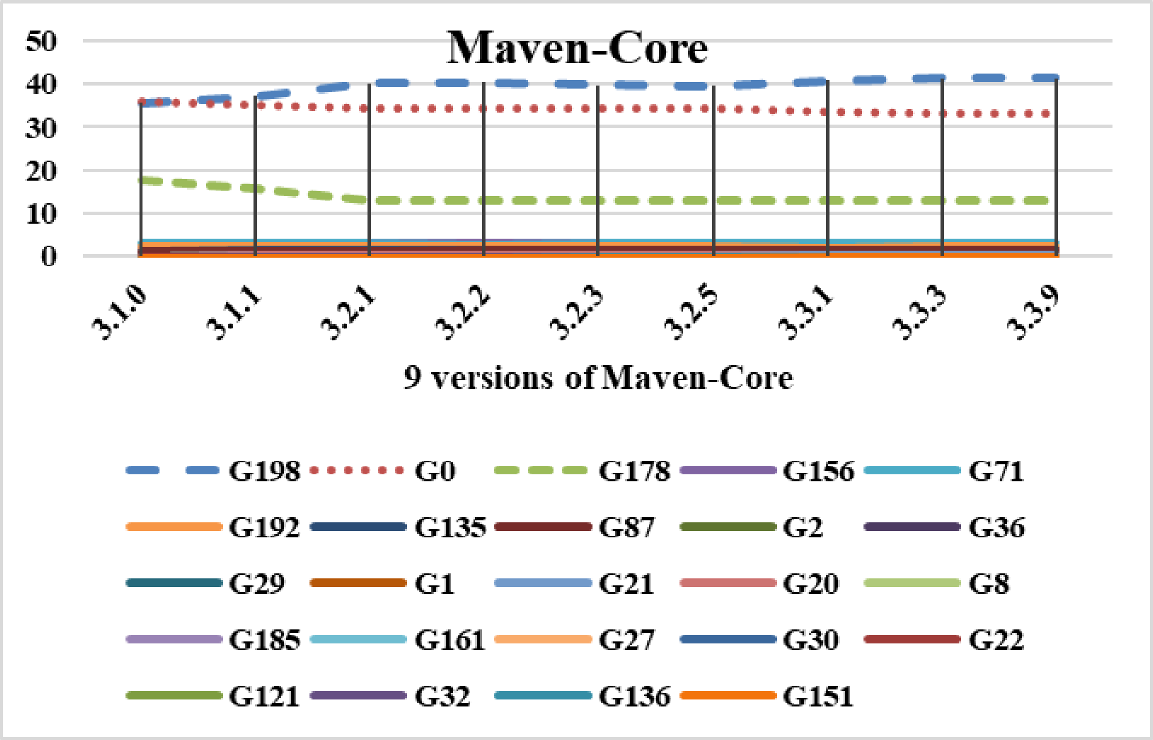
Complexities of each version and multiple versions

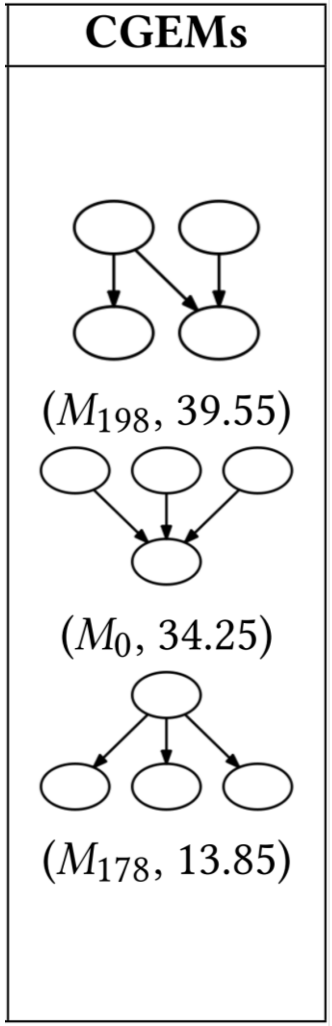
- Frequencies of CGEGs with their cyclomatic complexities are used to retrieve
  - varying complexity of each call graph (a varying time-series) of a version
  - an aggregated complexity of all evolving call graphs (a constant time-series over multiple versions).



Inferences

- Recurring subgraph patterns with low complexity occur in high percentage over a version series.
- These subgraphs are the CGEMs $\{M_{198}, M_0, M_{178}\}$ with cyclomatic complexity $= 1$, which has high frequencies of CGEGs $\{G_{198}, G_0, G_{178}\}$.





Inferences

- Changes in an evolving software lead to several versions.
- Dependency patterns also evolve in the versioning process to represent a state of the evolution information.
- Aggregate the state information over a version series to achieve evolution information about a whole centralized repository.
- This evolution information is helpful to manage and track evolution happening in a version series of an evolving software.

Answer to Research Question

- For call graph evolution, three inferences with Lehman's law of software evolution (in 1980) [18].
- Two similar inferences
  1. An evolving software undergoes continuous upgrading to make better versions.
  2. The stability of an evolving software remains almost constant with time.
- One dissimilar inference
  1. The complexity of evolving software keeps on changing with time, but not necessarily increasing.
     - Because, now there are better software repository management techniques and systems (e.g. GitHub, Maven) are easily accessible as compared to the era when Lehman's law was introduced.

[18] M. M. Lehman. "Programs, life cycles, and laws of software evolution". *Proceedings of the IEEE* 68.9 (1980): 1060-1076.

Conclusions

Conclusions

- Introduced two proposed techniques,
  - analyze 10 evolving software systems (including Maven-Core described in detail)
- Looking forward to publishing a complete study of
  - Stable CGERs mining,
  - CGESs mining,
  - provide intrinsic details of the approaches,
  - demonstrate detailed study of 10 evolving software systems
- Author already published
  - System Evolution Analytics for Hadoop-HDFS repository
  - Cloud Service Evolution Analytics

Related Publications

Citation

Animesh Chaturvedi. 2022. Call Graph Evolution Analytics over a Version Series of an Evolving Software System. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA*. ACM, New York, NY, USA, 5 pages.

System Network Evolution Rules and Subgraphs

[34] Animesh Chaturvedi, Aruna Tiwari, and Nicolas Spyratos. "minStab: Stable Network Evolution Rule Mining for System Changeability Analysis". *IEEE Transactions on Emerging Topics in Computation Intelligence*, Vol 5.2 (April 2019). DOI: 10.1109/TETCI.2019.2892734.

[35] Animesh Chaturvedi, Aruna Tiwari, and Nicolas Spyratos. "System Network Analytics: Evolution and Stable Rules of a State Series." *IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2022.

[36] Animesh Chaturvedi and Aruna Tiwari. "System Network Complexity: Network Evolution Subgraphs of System State Series." *IEEE Transactions on Emerging Topics in Computational Intelligence,* Vol 4.2 (2018): 130-139. DOI: 10.1109/TETCI.2018.2848293. (IEEE Computer Society and IEEE Computational Intelligence Society).

[37] Animesh Chaturvedi and Aruna Tiwari. "System Evolution Analytics: Evolution and Change Pattern Mining of Inter-Connected Entities". *48th 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* pp. 3877-3882. IEEE SMC Society DOI: 10.1109/SMC.2018.00750.

System Neural Network and Graph Evolution Learning

[38] Animesh Chaturvedi and Aruna Tiwari. "System Evolution Analytics: Deep Evolution and Change Learning of Inter-Connected Entities". *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018: 3075-3080.

[39] Animesh Chaturvedi and Aruna Tiwari. "SysEvoRecomd: Graph Evolution and Change Learning based System Evolution Recommender". *IEEE International Conference on Data Mining Workshop (ICDMW)*, 2018, 1499-1500.

[40] Animesh Chaturvedi, Aruna Tiwari, and Shubhangi Chaturvedi. "SysEvoRecomd: Network Reconstruction by Graph Evolution and Change Learning". *IEEE Systems Journal* 14.3 (2020): 4007-4014.

[41] Animesh Chaturvedi, Aruna Tiwari, Shubhangi Chaturvedi, and Pietro Liò. "System Neural Network: Evolution and Change based Structure Learning". *IEEE Transactions on Artificial Intelligence* 3.3 (2022): 426-435.

Service Evolution Analytics

[42] Animesh Chaturvedi, Aruna Tiwari, Shubhangi Chaturvedi, and Dave Binkley "Service Evolution Analytics: Change and Evolution Mining of a Distributed System", *IEEE Transactions on Engineering Management*, Vol. 68.1, pp 137 - 148, Feb-2021, DOI: 10.1109/TEM.2020.2987641 IF: 2.784. (ABDC A).

[43] Animesh Chaturvedi, and Dave Binkley. "Web Service Slicing: Intra and Inter-Operational Analysis to Test Changes". *IEEE Transactions on Services Computing* Vol. 14.3 (May-June 2021): 930-943. IF: 4.91 DOI: 10.1109/TSC.2018.2821157 (CORE A*).

[44] Animesh Chaturvedi, "Subset WSDL to access Subset Service for Analysis", *6th IEEE International Conference on Cloud Computing Technology and Science* (IEEE CloudCom), Singapore, Dec 2014, pp 688-691. *IEEE Computer Society* and *IEEE Cloud Computing* DOI: 10.1109/CloudCom.2014.149.

[45] Animesh Chaturvedi, "Automated Web Service Change Management AWSCM - A Tool" *6th IEEE International Conference on Cloud Computing Technology and Science* (IEEE CloudCom), Singapore 2014, pp 715-718. *IEEE Computer Society* and *IEEE Cloud Computing* DOI: 10.1109/CloudCom.2014.144.

Acknowledgments

Thanks to

Prof. Aruna Tiwari
Indian Institute of Technology
Indore, India

Prof. Dave Binkley
Loyola University Maryland
Baltimore, USA

Prof. Nicolas Spyratos
University of Paris-Saclay
Paris, France

ขอบคุณ

Thai

Grazie

Italian

תודה רבה

Hebrew

धन्यवादः

Sanskrit

ಧನ್ಯವಾದಗಳು

Kannada

Ευχαριστώ

Greek

Thank You

English

Gracias

Spanish

Спасибо

Russian

Obrigado

Portuguese

شكراً

Arabic

https://sites.google.com/site/animeshchaturvedi07

Merci

French

多謝

Traditional Chinese

धन्यवाद

Hindi

Danke

German

நன்றி

Tamil

Tamil

多谢

Simplified Chinese

ありがとうございました

Japanese

감사합니다

Korean