PENGGUNAAN PROGRAM CKJM UNTUK ANALISIS PAKET *REMOTE METHOD INVOCATION*

Adi Kusjani, Berta Bednar

Jurusan Teknik Komputer
STMIK Akakom
Yogyakarta

adikusja@akakom.ac.id

Abstrak

Sistem Remote Method Invocation (RMI) memungkinkan sebuah objek yang berjalan dalam suatu Java Virtual Machine (JVM) untuk memanggil metode pada objek yang berjalan di JVM yang lain. RMI menyediakan untuk komunikasi jarak jauh antara program yang ditulis dalam bahasa pemrograman Java. Dalam pemrograman client/server, yang ditulis dalam bahasa Java, menawarkan RMI sebagai alternative dari teknologi system terdistribusi dan merupakan sebuah teknik pemanggilan metode terpisah yang menggunakan paradigma Object Oriented Programming (OOP).

Penelitian ini menganalisis source code kelas-kelas java pada paket RMI dengan metode CK-Metrics ,menggunakan program CKJM (Chidamber and Kemerer Java Metrics) yang direkomendasikan untuk menghasilkan matrik Chidamber and Kemerer bagi bahasa berorientasi objek sebagai sarana menganalisis kualitas perangkat lunak menggunakan kaidah-kaidah: reusability, understandability, maintainability dan testability.

Hasil analisis menggunakan program CKJM pada kelas-kelas pada paket RMI menunjukkan tingkat kaidah-kaidah reusability, understandability, maintainability dan testability yang baik, dengan catatan: khusus kelas RemoteException nilai NOC=12, berarti menunjukkan tingkat reusability dan testability tidak baik.

Kata Kunci: CKJM, *CK-Metrics*, *maintainability*, *reusability*, RMI, sistem terdistribusi, *testability*, *understandability*.

1. Pendahuluan

1.1 Latar Belakang

Sistem *Remote Method Invocation* (RMI) memungkinkan sebuah objek yang berjalan dalam suatu Java Virtual Machine (JVM) untuk memanggil metode pada objek yang berjalan di Java Virtual Machine yang lain. RMI menyediakan untuk komunikasi jarak jauh antara program yang ditulis dalam bahasa pemrograman Java. Teknologi informasi saat ini cenderung mengarah pada sistem terdistribusi, di mana fungsi sebuah komputer tidak hanya terbatas pada mesin itu sendiri, tetapi juga dapat digunakan secara terkoordinasi dengan banyak komputer. Secara umum, semua mesin yang terhubung ke internet dikategorikan dalam dua tipe, yaitu: *server* (mesin yang memberikan layanan pada mesin lain) dan *client* (mesin yang meminta layanan pada mesin lain). Pada

pemrograman *client/server*, *client* dan *server* dapat ditulis dalam bahasa yang berbeda, Java menawarkan RMI sebagai alternatif dari teknologi sistem terdistribusi dan merupakan sebuah teknik pemanggilan metode terpisah.

CKJM (*Chidamber and Kemerer Java Metrics*) digunakan sebagai *tool* yang telah direkomendasikan untuk menghasilkan matriks berorientasi objek sebagai sarana guna menganalisis kualitas perangkat lunak yang baik. CKJM tidak berbasis GUI yang fokusnya pada diagram yang rumit tetapi pada perhitungan metrik yang efisien. CKJM merupakan proyek *open source* dan gratis, dan juga dapat dikatakan sebagai *tool* yang sangat layak untuk digunakan untuk mengukur kualitas perangkat lunak berbasis objek dengan metode CK-Metrics. Makalah ini akan menyajikan hasil pengukuran kualitas kelas-kelas paket RMI di Java, menggunakan CKJM versi 1.9 berupa metrik sebagai indikator kualitas yang baik (Jureczko dan Spinellis, 2010).

1.2 Batasan Masalah

Batasan dari penelitian ini adalah sebagai berikut:

1. Kaidah-kaidah yang menunjukkan kualitas dari kelas-kelas paket RMI meliputi: *reusability*, *understandability*, *maintainability*, dan *testability*.
2. Metode untuk menganalisa dengan CK-Metrics yang menggunakan program CKJM.

1.3 Manfaat Penelitian

Manfaat dari penelitian ini adalah untuk mengetahui kualitas kelas-kelas pada paket RMI, yang dapat dijadikan masukkan bagi pembuat aplikasi dalam membangun aplikasi client-server yang menggunakan paket RMI.

2. Metode Penelitian

Metode yang digunakan dalam menyelesaikan penelitian ini dapat dijabarkan sebagai berikut:

1. Menginstal program CKJM pada komputer, dimana program ini akan digunakan untuk mengukur kelas-kelas pada paket RMI dengan beberapa *method* CK-Metrics, dengan ketentuan sebagai berikut:
    - WMC, menghitung jumlah *method* yang diimplementasikan dalam kelas.
    - DIT, menghitung jumlah tingkatan dari kelas *node* ke *root* dari *inheritance hierarchy tree*.

- NOC, menghitung jumlah *subclass* yang diturunkan langsung dari suatu *class*.
- CBO, menjumlahkan *class* lainnya yang *non-inheritance* dimana *class* tersebut di *couple* (di dalam satu *class* memanggil method dari *class* lainnya).
- RFC, menghitung banyaknya *method* yang kemungkinan dieksekusi sebagai *response* atas *message* objek dari kelas tersebut.
- LCOM, menghitung jumlah *method* yang tidak memiliki irisan atribut dengan *method* lainya dikurangi dengan *method* yang memiliki irisan atribut dengan *method* lainnya.

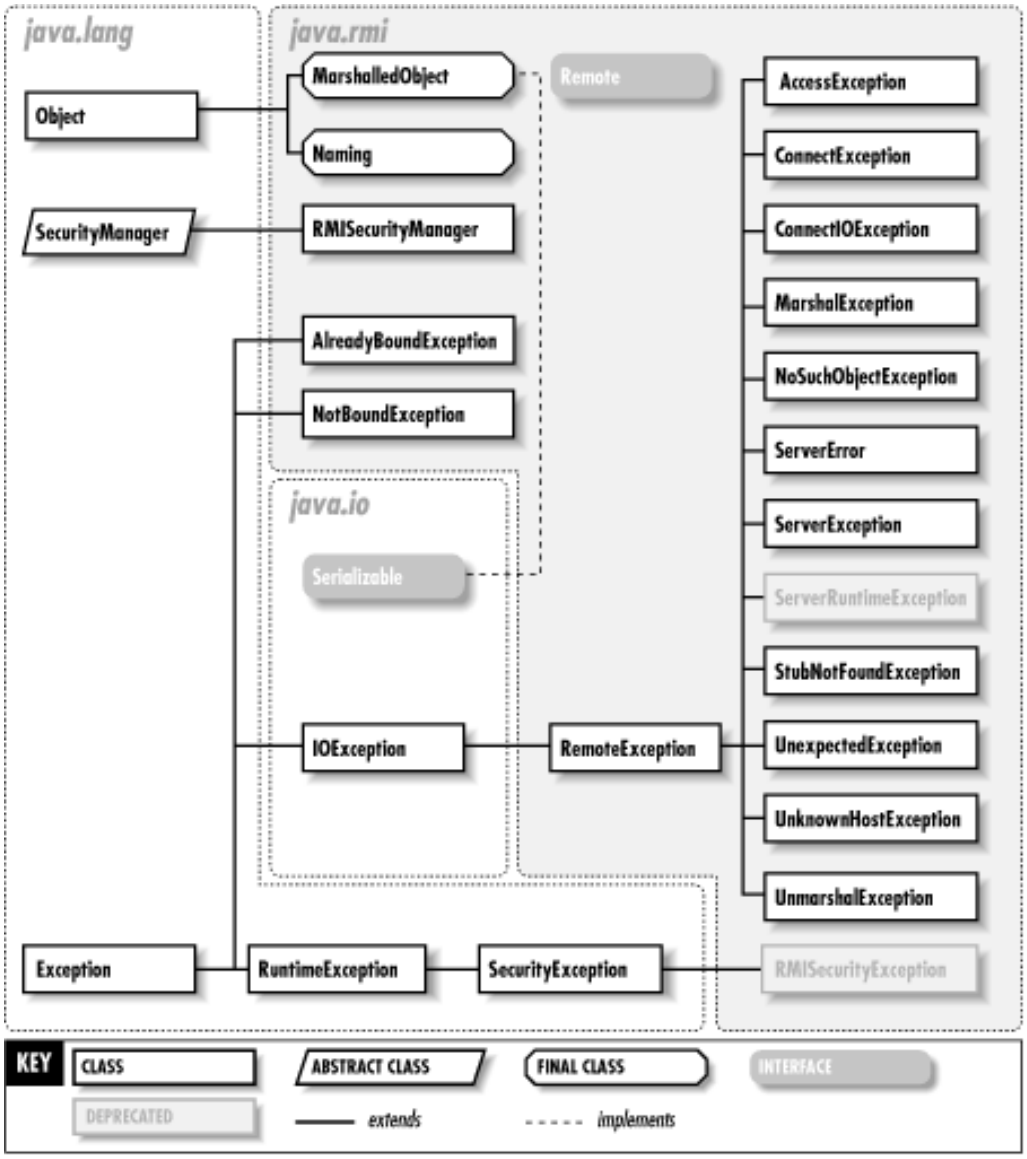
2. Menganalisis hasil pengukuran kelas-kelas paket RMI mengunakan beberapa *method* CK-Metrics, dengan kaidah-kaidah untuk menganalisa meliputi: *reusability*, *understandability*, *maintainability* dan *testability*, berdasarkan tabel matriks yang didapatkan.

3. Pengujian dan Pembahasan

3.1 Paket RMI (*Remote Method Invocation*)

Paket RMI menyediakan sarana dimana *client* dan *server* dapat berkomunikasi dan saling bertukar infomasi. RMI memungkinkan pengembang perangkat lunak untuk merancang aplikasi terdistribusi dimana *method* dari *remote object* dapat dipanggil dari JVM (Java Virtual Machine) lain, yang mungkin berjalan pada *host* yang berbeda. *Remote object* adalah obyek dalam Java yang dapat direferensikan secara *remote*. Pemrogram seolah memanggil *method* lokal dari file kelas lokal, sedang dalam kenyataannya semua argumen dikirimkan ke *remote target* dan diinterpretasikan, kemudian hasilnya dikirimkan kembali ke pemanggil.

Paket java.rmi adalah paket utama RMI, yang mengandung objek-objek prinsip yang digunakan dalam klien dan server RMI. Gambar 1 menunjukkan hirarki kelas untuk paket java.rmi ini.



Gambar 1 Hirarki kelas untuk paket java.rmi (Flanagan, et al., 1999)

3.2 Analisis dengan Metrik WMC

WMC dihitung berdasarkan banyaknya method suatu kelas. Tabel 1 menunjukkan tabel hasil perhitungan WMC.

Dari hasil perhitungan WMC, diperoleh bahwa nilai minimal WMC adalah 0, nilai maksimal WMC adalah 14 dan nilai rata-rata WMC adalah 4,34. Dengan ketentuan dimana nilai WMC baik bilamana maksimal antara 20 sampai 50 dan dapat memiliki nilai WMC > 24 dengan catatan jumlah kelas yang memiliki nilai WMC > 24 maksimal 10% (Aviosto Oy, n.d.). Ini menunjukkan kelas-kelas paket java.rmi memiliki nilai WMC yang baik, maka dapat disimpulkan pengukuran dengan metrik WMC pada kelas-kelas paket java.rmi menunjukkan tingkat *understandability*, *maintainability*, *reusability* dan *testability* yang baik.

Tabel 1 Hasil Perhitungan WMC dari Kelas-Kelas Paket RMI

Nama Kelas	WMC	Nama Kelas	WMC	Nama Kelas	WMC	Nama Kelas	WMC
AccessException	2	UnexpectedException	2	DGC	2	RMIClassLoader	14
AlreadyBoundException	2	UnknownHostException	2	Lease	3	RMIClassLoaderSpi	5
ConnectException	2	UnmarshalException	2	VMID	6	RMIClientSocketFactory	1
ConnectIOException	2	Activatable	14	LocateRegistry	8	RMIFailureHandler	1
MarshalException	2	ActivateFailedException	2	Registry	5	RMIServerSocketFactory	1
MarshalledObject	4	ActivationDesc	11	RegistryHandler	2	RMISocketFactory	9
Naming	9	ActivationException	5	ExportException	2	ServerCloneException	4
NoSuchObjectException	1	ActivationGroup	14	LoaderHandler	3	ServerNotActiveException	2
NotBoundException	2	ActivationGroupDesc	9	LogStream	11	ServerRef	2
Remote	0	ActivationGroupID	4	ObjID	10	Skeleton	2
RemoteException	5	ActivationID	6	Operation	3	SkeletonMismatchException	1
RMISecurityException	2	ActivationInstantiator	1	RemoteCall	7	SkeletonNotFoundException	2
RMISecurityManager	1	ActivationMonitor	3	RemoteObject	9	SocketSecurityException	2
ServerError	1	ActivationSystem	10	RemoteObjectInvocationHandler	8	UID	9
ServerException	2	Activator	1	RemoteRef	8	UnicastRemoteObject	11
ServerRuntimeException	1	UnknownGroupException	1	RemoteServer	6	Unreferenced	1
StubNotFoundException	2	UnknownObjectException	1	RemoteStub	3		
SubTotal	**40**		**88**		**96**		**67**
Total	**291**						

3.3 Analisis *Depth of Inheritance Tree* (DIT)

DIT diukur berdasarkan jumlah dari induk *class*-nya. Pada Tabel 2, ditunjukkan tabel hasil perhitungan DIT.

Tabel 2 Hasil Perhitungan DIT dari Kelas-Kelas Paket RMI

Nama Kelas	DIT	Nama Kelas	DIT	Nama Kelas	DIT	Nama Kelas	DIT
AccessException	5	UnexpectedException	5	DGC	1	RMIClassLoader	1
AlreadyBoundException	3	UnknownHostException	5	Lease	1	RMIClassLoaderSpi	1
ConnectException	5	UnmarshalException	5	VMID	1	RMIClientSocketFactory	1
ConnectIOException	5	Activatable	3	LocateRegistry	1	RMIFailureHandler	1
MarshalException	5	ActivateFailedException	5	Registry	1	RMIServerSocketFactory	1
MarshalledObject	1	ActivationDesc	1	RegistryHandler	1	RMISocketFactory	1
Naming	1	ActivationException	3	ExportException	5	ServerCloneException	4
NoSuchObjectException	5	ActivationGroup	4	LoaderHandler	1	ServerNotActiveException	3
NotBoundException	3	ActivationGroupDesc	1	LogStream	4	ServerRef	1
Remote	1	ActivationGroupID	1	ObjID	1	Skeleton	1
RemoteException	4	ActivationID	1	Operation	1	SkeletonMismatchException	5
RMISecurityException	5	ActivationInstantiator	1	RemoteCall	1	SkeletonNotFoundException	5
RMISecurityManager	2	ActivationMonitor	1	RemoteObject	1	SocketSecurityException	6
ServerError	5	ActivationSystem	1	RemoteObjectInvocationHandler	2	UID	1
ServerException	5	Activator	1	RemoteRef	1	UnicastRemoteObject	3
ServerRuntimeException	5	UnknownGroupException	4	RemoteServer	2	Unreferenced	1
StubNotFoundException	5	UnknownObjectException	4	RemoteStub	2		
SubTotal	**65**		**46**		**27**		**36**
Total	**174**						

Dari hasil perhitungan DIT, diperoleh bahwa nilai minimal DIT adalah 1, nilai maksimal DIT adalah 6 dan nilai rata-rata DIT adalah 2,59. Disarankan nilai DIT <= 5, karena jika memiliki nilai yang berlebihan maka akan menjadi lebih komplek untuk dikembangkan (Aviosto Oy, n.d.). Hal ini menunjukkan kelas-kelas paket java.rmi memiliki nilai DIT yang baik, maka disimpulkan pengukuran

dengan metrik DIT pada kelas-kelas paket java.rmi menunjukkan tingkat *understandability* yang baik, sedangkan tingkat *reuseability* tidak baik karena kedalamannya kurang.

3.4 Analisis *Number of Children* (NOC)

NOC diukur berdasarkan jumlah *subclass* yang diturunkan langsung dari suatu *class*. Pada Tabel 3, ditunjukkan tabel hasil perhitungan NOC.

Tabel 3 Hasil Perhitungan NOC dari Kelas-Kelas Paket RMI

Nama Kelas	NOC	Nama Kelas	NOC	Nama Kelas	NOC	Nama Kelas	NOC
AccessException	0	UnexpectedException	0	DGC	0	RMIClassLoader	0
AlreadyBoundException	0	UnknownHostException	0	Lease	0	RMIClassLoaderSpi	1
ConnectException	0	UnmarshalException	0	VMID	0	RMIClientSocketFactory	0
ConnectIOException	0	Activatable	0	LocateRegistry	0	RMIFailureHandler	0
MarshalException	0	ActivateFailedException	0	Registry	0	RMIServerSocketFactory	0
MarshalledObject	0	ActivationDesc	0	RegistryHandler	0	RMISocketFactory	0
Naming	0	ActivationException	2	ExportException	1	ServerCloneException	0
NoSuchObjectException	0	ActivationGroup	0	LoaderHandler	0	ServerNotActiveException	0
NotBoundException	0	ActivationGroupDesc	0	LogStream	0	ServerRef	0
Remote	0	ActivationGroupID	0	ObjID	0	Skeleton	0
RemoteException	12	ActivationID	0	Operation	0	SkeletonMismatchException	0
RMISecurityException	0	ActivationInstantiator	0	RemoteCall	0	SkeletonNotFoundException	0
RMISecurityManager	0	ActivationMonitor	0	RemoteObject	3	SocketSecurityException	0
ServerError	0	ActivationSystem	0	RemoteObjectInvocation Handler	0	UID	0
ServerException	0	Activator	0	RemoteRef	0	UnicastRemoteObject	0
ServerRuntimeException	0	UnknownGroupException	0	RemoteServer	1	Unreferenced	0
StubNotFoundException	0	UnknownObjectException	0	RemoteStub	0		
SubTotal	**12**		**2**		**5**		**1**
Total	**20**						

Dari hasil perhitungan NOC, diperoleh bahwa nilai minimal NOC adalah 0, nilai maksimal NOC adalah 12 dan nilai rata-rata NOC adalah 0,29. Direkomendasikan oleh RefactorIT nilai NOC dibatasi antara 0 sampai 10, dan jika lebih dari 10, menunjukkan penyalahgunaan *subclassing* (Brooks, n.d.). Hal ini menunjukkan kelas-kelas paket java.rmi memiliki nilai NOC yang baik. Maka dapat disimpulkan pengukuran dengan metrik NOC pada kelas-kelas paket java.rmi menunjukkan tingkat *reusability* dan *testability* yang baik. Khusus kelas `RemoteException` nilai NOC adalah 12, menunjukkan adanya penyalahgunaan *subclassing*.

3.5 Analisis *Coupling Between Object* (CBO)

CBO menghitung jumlah *class* lainnya yang *non-inheritance* dimana *class* tersebut di-*couple*. *Couple* yang dimaksud adalah di dalam satu *class* memanggil *method* dari *class* lainnya. Tabel 4 menunjukkan tabel hasil perhitungan CBO.

Dari hasil perhitungan CBO, akan didapatkan nilai minimal CBO adalah 0, nilai maksimal CBO adalah 7 dan nilai rata-rata CBO adalah 0,32. Dikatakan oleh

Sahraoui, et al. (n.d.), nilai CBO > 14 menunjukkan nilai yang terlalu tinggi. Hal ini menunjukkan kelas-kelas pada paket java.rmi memiliki nilai CBO yang baik, maka dapat disimpulkan pengukuran dengan metrik CBO pada kelas-kelas dalam paket java.rmi menunjukkan tingkat *maintainability* dan *testability* yang baik.

Tabel 4 Hasil Perhitungan CBO dari Kelas-Kelas Paket RMI

Nama Kelas	CBO	Nama Kelas	CBO	Nama Kelas	CBO	Nama Kelas	CBO
AccessException	0	UnexpectedException	0	DGC	0	RMIClassLoader	1
AlreadyBoundException	0	UnknownHostException	0	Lease	0	RMIClassLoaderSpi	0
ConnectException	0	UnmarshalException	0	VMID	0	RMIClientSocketFactory	0
ConnectIOException	0	Activatable	3	LocateRegistry	7	RMIFailureHandler	0
MarshalException	0	ActivateFailedException	0	Registry	0	RMIServerSocketFactory	0
MarshalledObject	0	ActivationDesc	0	RegistryHandler	0	RMISocketFactory	1
Naming	0	ActivationException	0	ExportException	0	ServerCloneException	0
NoSuchObjectException	0	ActivationGroup	1	LoaderHandler	0	ServerNotActiveException	0
NotBoundException	0	ActivationGroupDesc	0	LogStream	0	ServerRef	0
Remote	0	ActivationGroupID	0	ObjID	1	Skeleton	0
RemoteException	0	ActivationID	0	Operation	0	SkeletonMismatchException	0
RMISecurityException	0	ActivationInstantiator	0	RemoteCall	0	SkeletonNotFoundException	0
RMISecurityManager	0	ActivationMonitor	0	RemoteObject	2	SocketSecurityException	0
ServerError	0	ActivationSystem	0	RemoteObjectInvocation Handler	0	UID	0
ServerException	0	Activator	0	RemoteRef	0	UnicastRemoteObject	3
ServerRuntimeException	0	UnknownGroupException	0	RemoteServer	3	Unreferenced	0
StubNotFoundException	0	UnknownObjectException	0	RemoteStub	0		
SubTotal	**0**		**4**		**13**		**5**
Total	**22**						

3.6 Analisis *Response for Class* (RFC)

RFC dihitung dari jumlah *method* lokal dan *method* yang dipanggil oleh *method* lokal termasuk semua *method* dalam kelas hirarki dan juga termasuk kelas *library*. Method yang sama dihitung sekali. Pada Tabel 5, ditunjukkan tabel hasil perhitungan RFC.

Dari hasil perhitungan RFC, didapatkan nilai minimal RFC adalah 0, nilai maksimalnya adalah 45 dan nilai rata-rata RFC adalah 10,61. Direkomendasikan oleh RefactorIT nilai ambang standar RFC yaitu 0 sampai 50 (Brooks, n.d.). Ini menunjukkan kelas-kelas pada paket java.rmi memiliki nilai RFC yang baik. Maka dapat disimpulkan pengukuran dengan RFC menunjukkan tingkat *reuseability*, *maintainability*, dan *testability* yang baik.

Tabel 5 Hasil Perhitungan RFC dari Kelas-Kelas Paket RMI

Nama Kelas	RFC	Nama Kelas	RFC	Nama Kelas	RFC	Nama Kelas	RFC
AccessException	4	UnexpectedException	4	DGC	2	RMIClassLoader	45
AlreadyBoundException	4	UnknownHostException	4	Lease	4	RMIClassLoaderSpi	6
ConnectException	4	UnmarshalException	4	VMID	21	RMIClientSocketFactory	1
ConnectIOException	4	Activatable	29	LocateRegistry	20	RMIFailureHandler	1
MarshalException	4	ActivateFailedException	4	Registry	5	RMIServerSocketFactory	1
MarshalledObject	15	ActivationDesc	20	RegistryHandler	2	RMISocketFactory	14
Naming	43	ActivationException	13	ExportException	4	ServerCloneException	11
NoSuchObjectException	2	ActivationGroup	45	LoaderHandler	3	ServerNotActiveException	4
NotBoundException	4	ActivationGroupDesc	18	LogStream	42	ServerRef	2
Remote	0	ActivationGroupID	9	ObjID	30	Skeleton	2
RemoteException	13	ActivationID	36	Operation	4	SkeletonMismatchException	2
RMISecurityException	3	ActivationInstantiator	1	RemoteCall	7	SkeletonNotFoundException	4
RMISecurityManager	2	ActivationMonitor	3	RemoteObject	35	SocketSecurityException	4
ServerError	2	ActivationSystem	10	RemoteObjectInvocation Handler	42	UID	28
ServerException	4	Activator	1	RemoteRef	8	UnicastRemoteObject	20
ServerRuntimeException	2	UnknownGroupException	2	RemoteServer	11	Unreferenced	1
StubNotFoundException	4	UnknownObjectException	2	RemoteStub	6		
SubTotal	**114**		**205**		**246**		**146**
Total	**711**						

3.7 Analisis *Lack of Cohesion in Methods* (LCOM)

LCOM digunakan mengukur derajat kemiripan *method* oleh variabel input data atau atribut dalam *class*. Tabel 6 menunjukkan hasil perhitungan LCOM.

Tabel 6 Hasil Perhitungan LCOM dari Kelas-Kelas Paket RMI

Nama Kelas	LCOM	Nama Kelas	LCOM	Nama Kelas	LCOM	Nama Kelas	LCOM
AccessException	1	UnexpectedException	1	DGC	1	RMIClassLoader	43
AlreadyBoundException	1	UnknownHostException	1	Lease	0	RMIClassLoaderSpi	10
ConnectException	1	UnmarshalException	1	VMID	1	RMIClientSocketFactory	0
ConnectIOException	1	Activatable	71	LocateRegistry	28	RMIFailureHandler	0
MarshalException	1	ActivateFailedException	1	Registry	10	RMIServerSocketFactory	0
MarshalledObject	0	ActivationDesc	19	RegistryHandler	1	RMISocketFactory	24
Naming	36	ActivationException	4	ExportException	1	ServerCloneException	0
NoSuchObjectException	0	ActivationGroup	61	LoaderHandler	3	ServerNotActiveException	1
NotBoundException	1	ActivationGroupDesc	2	LogStream	31	ServerRef	1
Remote	0	ActivationGroupID	0	ObjID	0	Skeleton	1
RemoteException	4	ActivationID	0	Operation	0	SkeletonMismatchException	0
RMISecurityException	1	ActivationInstantiator	0	RemoteCall	21	SkeletonNotFoundException	1
RMISecurityManager	0	ActivationMonitor	3	RemoteObject	0	SocketSecurityException	1
ServerError	0	ActivationSystem	45	RemoteObjectInvocation Handler	26	UID	0
ServerException	1	Activator	0	RemoteRef	28	UnicastRemoteObject	49
ServerRuntimeException	0	UnknownGroupException	0	RemoteServer	9	Unreferenced	0
StubNotFoundException	1	UnknownObjectException	0	RemoteStub	3		
SubTotal	**49**		**209**		**163**		**131**
Total	**552**						

Dari hasil perhitungan LCOM, didapatkan nilai minimal LCOM adalah 0, nilai maksimal LCOM adalah 71 dan nilai rata-rata RFC adalah 8,23. Sebagaimana direkomendasikan oleh NASA nilai rata-rata untuk LCOM pada CK-Metrics dengan kualitas "*low*" adalah 447,65, untuk kualitas "*medium*" adalah 113,94, dan untuk kualitas "*high*" adalah 78,34 (Aivosto Oy, n.d.). Hal ini menunjukkan kelas-kelas paket java.rmi memiliki nilai LCOM yang baik ("*high*").

Maka dapat disimpulkan pengukuran dengan metrik LCOM, menunjukkan tingkat *reuseability*, *maintainability* dan *understandability* yang baik.

4. Kesimpulan

Kesimpulan yang dapat diambil dari penelitian ini adalah sebagai berikut:

1. Pengukuran dengan metode metrik WMC, CBO, RFC, dan LCOM pada paket java.rmi menunjukkan tingkat *understandability*, *maintainability*, *reusability* dan *testability* yang baik.
2. Pengukuran dengan metrik NOC pada paket java.rmi menunjukkan tingkat *reusability* dan *testability* yang baik. Khusus kelas `RemoteException` nilai NOC sama dengan 12, berarti menunjukkan kemungkinan adanya penyalahgunaan *subclassing*.
3. Pengukuran dengan metrik DIT pada paket java.rmi menunjukkan tingkat *understandability* yang baik, sedangkan tingkat *reusability* kebalikannya karena kedalamannya kurang.

Daftar Pustaka

Aivosto Oy, n.d. *Project Metrics Help - Chidamber & Kemerer object-oriented metrics suite*. [Online] Available at: http://www.aivosto.com/project/help/pm-oo-ck.html [Accessed 12/05/2014].

Brooks, A., n.d. *Response for Class (RFC): Number of Distinct Methods and Constructors invoked by a Class*. [Online] Available at: http://staff.unak.is/andy/StaticAnalysis0809/metrics/rfc.html [Accessed 16/06/2014].

Brooks, A., n.d. *Number of Children in Tree (NOC): Number of Direct Subclasses of a Class*. [Online] Available at: http://staff.unak.is/andy/StaticAnalysis0809/metrics/noc.html [Accessed 12/06/2014].

Flanagan, D., Farley, J., Crawford, W. & Magnusson, K., 1999. *Java™ Enterprise in a Nutshell: A Desktop Quick Reference*. Sebastopol: O'Reilly Media.

Gray, N.A.B., 2005. Performance of Java Middleware – Java RMI, JAXRPC, and CORBA. *Proceedings of the Sixth Australasian Workshop on Software and System Architectures (AWSA 2005)*, pp. 31-39.

Jureczko, M. & Spinellis, D., 2010. Using Object-Oriented Design Metrics to Predict Software Defects. *Proceedings of RELCOMEX 2010: Fifth International Conference on Dependability of Computer Systems DepCoS, Monographs of System Dependability*, pp. 69-81.

Juric, M.B., Kezmah, B., Hericko, M., Rozman, I. & Vezocnik, I., 2004. Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis. *ACM SIGPLAN Notices*, 39(5), pp 58-65.

Sahraoui, H.A., Godin, R. & Miceli, T., n.d: *Can Metrics Help Bridging the Gap Between the Improvement of OO Design Quality and Its Automation?* [Online] Available at: http://www.iro.umontreal.ca/~sahraouh/papers/ICSM00.pdf [Accessed 17/12/2014].

Spinellis, D., 2005. Tool writing: A forgotten art? *IEEE Software*, 22(4), pp. 9-11.