Como Software Evolui? – Uma Abordagem de Redes

Kecia A. M. Ferreira, Mariza A. S. Bigonha, Roberto S. Bigonha, Bárbara Malta Gomes
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais(UFMG)
Belo Horizonte, Brasil
Email: {kecia,mariza,bigonha,barbara}@dcc.ufmg.br

***Resumo*—Conhecer como a evolução de software ocorre é de grande importância para o desenvolvimento e gestão de software. Isso tem motivado muitos trabalhos nas últimas décadas, cujos resultados apontam para o fato de que software cresce e sofre modificações continuamente, tem sua complexidade aumentada e sua qualidade deteriorada ao longo do tempo. Porém, o conhecimento sobre como esse processo ocorre ainda não está consolidado. Buscando avançar neste conhecimento, este trabalho analisa a evolução de software sob o olhar de redes. Um software pode ser modelado como uma rede, cujos nós representam as partes constituintes do software e as arestas, as relações entre essas partes. O estudo foi realizado com um conjunto 16 programas abertos desenvolvidos em Java. Os resultados do estudo revelam importantes fatos a respeito da natureza evolutiva desse tipo de software: o software cresce drástica e continuamente, sua densidade tende a diminuir, o diâmetro da rede cresce lentamente, as classes com maior grau de entrada tendem a se manter nessa situação durante toda a vida do software e têm sua qualidade deteriorada.**

***Abstract*—Knowledge of software evolution has a great importance for software development and management. The study of software evolution has been the subject of much researches in the last decades, whose results reveal that a software system has continuing growth, continuing changes, increasing complexity and declining quality. However, the knowledge about how this process occurs is not consolidate yet. The present work provides a better understanding of software evolution process by analising it through the view of networks. A software system can be modeled as a graph whose nodes represent the software modules and whose edges represent the relationships between them. The study was performed on a set of 16 open source software sytems developed in Java. The results of this study reveal importants facts about the evolutive nature of this type of software: software system grows drastically and continually, has decreasing density, its diameter grows slowly and its classes with large in-degree tend to keep this property and they have declining quality.**

***Keywords*-software evolution; object-oriented software; complex networks;**

I. Introdução

O fenômeno conhecido como deterioração de software é um problema clássico na Engenharia de Software. Apesar de todo o conhecimento produzido até então sobre como construir software de alta qualidade, sabe-se que ao longo de sua vida, um software passa por manutenções que possivelmente introduzem defeitos nele, tornam a sua estrutura mais complexa e rígida e degradam sua qualidade. As conhecidas Leis de Lehman [1] descrevem essa natureza da evolução de software, postulando que, dentre outros aspectos, software cresce e sofre manutenções continuamente, tem sua complexidade aumenta e sua qualidade diminuída ao longo do tempo.

Conhecer o processo de evolução de software tem importância central para a compreensão das leis que regem a deterioração de software e isso tem motivado muitas pesquisas. A maior parte dos trabalhos realizados com esse objetivo avaliam a evolução do software por meio de métricas de software, tais como tamanhos dos sistemas de software e número de modificações realizadas neles ao longo de suas vidas [2], [3]. Por exemplo, um estudo recente de Xie et al. [4] investigou a aplicabilidade das leis de Lehman em software aberto, concluindo que esse tipo de software cresce e sofre modificações continuamente e tem sua complexidade aumentada. O estudo, contudo, não confirma que a qualidade desse tipo de software declina ao longo do tempo. Outra conclusão importante desse trabalho é que verificou-se que um alto percentual de modificações concentram-se em um pequeno percentual de partes do código fonte do programa. Embora as pesquisas realizadas até então sobre evolução de software tenham chegado a conclusões bastante significativas, ainda é necessário um olhar mais detalhado sobre esse processo evolutivo.

Uma forma de observar um sistema de software é representá-lo como uma rede na qual os nós são os módulos do software - suas partes constituintes - e as arestas são os relacionamentos entre essas partes. Os processos de evolução de muitas redes reais têm sido estudados, tais como a Internet, a Web e redes sociais [5], [6]. Os resultados dos estudos contribuem de forma determinante para a conhecimento das características e evolução dessas redes e para a predição do comportamento delas. Esse tipo de análise responde questões como: quais são os nós da rede mais importantes para manter sua conectividade, com o que a rede se parece quando visualizada graficamente e como a rede cresce ao longo do tempo.

Neste trabalho, é conduzido um estudo empírico sobre evolução de software sob um olhar de redes. Os dados utilizados no estudo são de 16 sistemas de software abertos

orientados por objetos, totalizando 109 versões dos programas. O estudo tem por objetivo investigar as seguintes questões:

- Evolução do grau de entrada das classes: um sistema orientado por objetos pode ser representado por uma rede na qual as classes correspondem aos nós e os relacionamentos entre elas, às arestas. Neste estudo, identifica-se se classes com alto grau de entrada (*in-degree*) tendem a manter essa condição ao longo da vida do software.
- Evolução do tamanho do software: o estudo visa observar como o número de módulos - classes - do software cresce ao longo do tempo.
- Evolução da densidade da rede: a densidade da rede é dada pela razão entre o número de arestas existentes nela e o número máximo possível de arestas da rede [6]. O estudo observa como a densidade de um software comporta-se ao longo do tempo.
- Evolução do diâmetro da rede: *diâmetro* é uma métrica que corresponde ao maior caminho mínino entre os nós da rede. Esta métrica revela importantes características de uma rede. Por exemplo, redes do tipo *small-word* são caracterizadas por terem diâmetro pequeno e pelo crescimento pequeno do diâmetro com relação ao crescimento do número de nós da rede [5].
- Visualização gráfica da rede: essa visualização permite perceber qual é o desenho da rede formada pelos módulos de um software. Isso fornece, por exemplo, a informação de existência ou não de um componente gigante na rede. A existência de um componente gigante na rede indica que há uma grande porção de nós entre os quais a comunicação é possível [5]. No caso de software, indica que há uma grande porção de classes entre as quais há uso direto ou indireto.
- Estabilidade de classes com alto grau de entrada: identifica-se se classes com alto grau de entrada tendem a se manter estáveis ao longo da vida do software. O termo *estabilidade* aqui refere-se à situação em que a classe sofre poucas modificações.
- Evolução da coesão interna das classes: a coesão interna de uma classe é definida como o grau de relacionamento entre seus elementos internos. É desejável que uma classe possua alta coesão. Classes com baixa coesão implementam vários papéis no software e são de difícil compreensão. A coesão é um indicador da qualidade estrutural da classe. Esse estudo visa observar como a coesão interna das classes em um sistema evolui.

Os resultados dessa análise revelam importantes conhecimentos sobre as caracteríscticas de software e sua evolução. Este artigo está organizado da seguinte maneira: a Seção II especifica as métricas utilizadas na análise dos dados; a Seção III descreve a metodologia empregada no estudo; a Seção IV apresenta os resultados do trabalho; a Seção V apresenta trabalhos relacionados e na Seção VI são descritas as conclusões alcançadas com o estudo.

II. MÉTRICAS

Um software orientado por objetos pode ser modelado como um grafo direcionado - uma rede - no qual os nós correspondem às classes e os vértices são as conexões entre as classes. Neste trabalho considera-se que uma classe *A* está conectada a *B* se usa um atributo ou um método de *B*, ou se é herdeira de *B*. Neste caso, há uma aresta de *A* para *B*. Neste estudo, a evolução do software é analisada por meio de métricas de software e métricas de rede. Este seção descreve as métricas utilizadas.

- Densidade da rede: esta métrica é dada pela razão entre o número de arestas no grafo e o número máximo de arestas possíveis nele [6]. Em um grafo direcionado com n módulos e a arestas, a densidade é dada por $a/(n.(n-1))$. A métrica de software, denominada COF (*fator acoplamento*) [7], corresponde à essa densidade.

Tabela I
PROGRAMAS ANALISADOS NO ESTUDO

Nome	*Categoria*	*# downloads/ semana*	*Tempo de vida*	*#classes*	*#versões*	*#versões analisadas*
JEdit	Editor de texto	9.138	2001 a 2009	377 a 1124	13	13
Dr Java	Desenvolvimento	3.837	2002 a 2009	596 a 3692	10	10
Java Groups	Apoio à cooperação	465	2003 a 2009	696 a 1137	40	13
KoL Mafia	Jogo	1.007	2004 a 2009	39 a 1109	13	13
DBUnit	Banco de dados	448	2002 a 2009	198 a 369	25	5
FreeCol	Jogos	7.452	2003 a 2010	112 a 5902	27	5
JasperReports	Desenvolvimento de software	5.542	2001 a 2010	525 a 5304	50	5
JGNash	Financeiro	822	2002 a 2010	782 a 3603	40	5
Java msn library	Comunicação	271	2004 a 2010	494 a 872	10	5
Jsch	Segurança	2.304	2004 a 2009	202 a 271	29	5
JUnit	Desenvolvimento de software	1.834	2000 a 2009	78 a 230	18	5
Logisim	Educação	1.590	2005 a 2009	908 a 1185	28	5
MeD's Movie Manager	Armazenamento	1.169	2003 a 2010	64 a 517	60	5
Phex	Redes	1.084	2001 a 2009	393 a 1352	26	5
Squirrel sql	Banco de dados	7.270	2006 a 2010	424 a 1223	26	5
Hibernate	Banco de dados	12.906	2004 a 2010	956 a 2446	53	5

Essa métrica é um indicador de quão conectado o software está. Quanto maior o COF, mais interdependentes são os módulos do software e mais difícil sua manutenção. Entender como a densidade do software evolui é importante para se identificar como esse grau de interdependência entre os módulos se comporta como o crescimento do software.

- Diâmetro da rede: essa métrica de rede é dada pelo maior caminho mínimo entre os nós da rede [5]. No caso de software, o diâmetro da rede é um indicador do grau de impacto de modificações de classes. Por exemplo, em um software com diâmetro 5, uma modificação da classe que está no início do caminho pode gerar impacto em 4 outras classes no mínimo.
- Grau de entrada: é dada pelo total de classes que utilizam determinada classe. Assim como em outras redes, quanto maior o grau de entrada, maior a importância do nó na rede. Classes com alto grau de entrada são fornecedoras de serviços para um grande número de outras classes. Desta forma, erros ou manutenções nestas classes podem gerar um amplo impacto no sistema.
- Coesão interna de classe: há um grande número de métricas de coesão de classes propostas na literatura. Porém, ainda não há um consenso sobre a forma de se medir tal aspecto. A métrica mais amplamente conhecida é LCOM, que tem sido também muito criticada. Neste estudo, é utiliza uma métrica denominada Coesão de Interesse (CoIn) [8]. Esta métrica é dada por $1/C$, onde C é o número de conjuntos M_i disjuntos de métodos da classes; cada um desses conjuntos é constituído por métodos que têm similaridade entre si. Nesta métrica, um método é considerado similar ao outro se ambos utilizam variáveis de instância da classe ou métodos em comum. Por exemplo, se são identificados 2 conjuntos, a métrica de coesão interna resulta em 0,5. Isso é um indicador de que a classe implementa dois papéis; se é identificado somente um conjunto, a métrica resulta em 1, indicando que a classe implementa um único papel no sistema, o que é desejável.
- Número de métodos públicos da classe: corresponde ao total de métodos públicos definidos na classe. Essa métrica é utilizada para avaliar como os tamanhos das interfaces das classes evoluem ao longo do tempo.
- Número de atributos públicos: corresponde ao total de atributos definidos na classe. Essa métrica também é utilizada neste estudo para avaliar a evolução do tamanho das classes.

III. Metodologia

A seleção dos sistemas de software para o estudo foi realizada de acordo com os seguintes critérios: tempo de vida, quantidade de versões e categoria. O repositório de dados utilizado foi www.sourceforge.net, que classifica os programas em 13 categorias, tais como desenvolvimento, jogos e comunicação. Para cada categoria, foram selecionados até 10 sistemas de software desenvolvidos em Java com pelo menos 5 versões e com pelo menos 4 anos de vida. Outro aspecto observado é a disponibilidade dos arquivos compilados do programa *bytecodes* ou a facilidade de sua geração, pois a ferramenta utilizada na coleta das medidas das métricas avalia o arquivo compilado e não o código fonte do programa. O levantamento inicial resultou em 108 programas. Dentre os programas identificados, foram selecionados para análise, por categoria, aqueles com maior número de versões, mais tempo de vida e mais populares. Para avaliar popularidade, utilizou-se o número de *downloads* semanais do software. Essa última seleção resultou em 16 programas. A Tabela I mostra os dados dos programas analisados no estudo. Os dados foram obtidos de Sourceforge.net no período de Setembro de 2009 a Abril de 2010.

Para os quatro primeiros programas da Tabela I, foram analisadas todas as versões, exceto no caso de Java Groups, que possui um número grande de versões, dentre as quais foram selecionadas 13 delas: a primeira, a última e outras 11 intermediárias, observando-se um período de liberação entre as versões aproximadamente igual. Os resultados desse primeiro grupo de software foi analisado. Como percebeu-se que nos quatro casos os dados de versões subsequentes eram muito próximos, para os demais programas foram selecionadas 5 versões: a primeira, a última, e três intermediárias, com período entre as versões aproximadamente igual.

Para cada versão de cada software, foram coletadas as métricas de software com a ferramenta Connecta, desenvolvida pelo grupo de pesquisadores que realizaram este estudo. Esta ferramenta gera um arquivo que contém os dados do grafo que representa o software, compatível com o arquivo de entrada do Pajek [9], uma ferramenta de coleta de metricas de rede e geração de representações gráficas da rede. As métrica diâmetro da rede e as imagens da rede foram coletadas com Pajek.

Após a coleta das métricas, em cada versão, as classes foram ordenadas por número de conexões aferentes (grau de entrada), da maior para a menor. Identificadas as classes de maior grau de entrada nas versões, analisou-se a evolução dessas classes em termos das três métricas de software: coesão interna, número de métodos e de atributos públicos. Além disso, observou-se como o número de grau de entrada dessas classes comporta-se ao longo do tempo.

IV. Resultados

Esta seção descreve e analisa os resultados do estudo realizado neste trabalho. Para os programas JEdit, Dr Java, Java Groups e KolMafia, para os quais foram analisadas um grande número de versões, os dados são mostrados por meio de gráficos. Os dados dos demais programas são mostrados na Tabela II.

Tabela II
DADOS DE EVOLUÇÃO DOS PROGRAMAS ANALISADOS NO ESTUDO

Software	*Versão*	*#Classes*	*#Conexões*	*COF*	*Diâmetro*
DBUnit	2.0	198	429	0,011	9
	2.2.1	289	666	0,008	11
	2.4.0	332	769	0,007	13
	2.4.4	347	720	0,006	17
	2.4.7	369	815	0,006	16
FreeCol	0.1.0	44	112	0,05900	5
	0.5.0	416	1899	0,011	12
	0.6.0	611	2609	0,007	11
	0.8.0	927	5150	0,006	13
	0.9.2	1087	5902	0,005	14
Jasper Reports	0.4.0	242	525	0,009	8
	1.0.0	574	1316	0,004	9
	2.0.0	1104	2435	0,002	13
	3.0.0	1233	3038	0,002	13
	3.7.1	1629	5304	0,002	13
JGNash	1.10.0	743	2757	0,005	16
	1.11.1	782	2443	0,004	17
	1.50.0	942	2659	0,003	12
	2.00.0	2716	7374	0,001	24
	2.20.0	3603	12978	0,001	24
Java msn library	10a1	171	494	0,017	10
	10a2	186	516	0,015	7
	10b1	203	615	0,015	7
	10b2	218	662	0,014	9
	10b3	270	872	0,012	9
JUnit	3.4	78	138	0,023	5
	3.8	101	182	0,018	6
	4.0	92	167	0,02	6
	4.5	188	352	0,01	8
	4.8.1	230	421	0,008	10

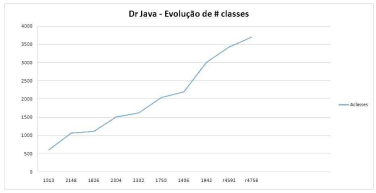
Software	*Versão*	*#Classes*	*#Conexões*	*COF*	*Diâmetro*
LogSim	2.0.0	908	3294	0,004	13
	2.1.0	993	3940	0,004	14
	2.1.5	1018	4141	0,004	14
	2.2.0	1054	4439	0,004	14
	2.3.3	1185	4209	0,003	14
MeD's Movie	1.6	64	149	0,037	6
Manager	1.7	73	168	0,032	6
	2.8	458	1465	0,007	12
	2.9.13	608	1845	0,005	13
	2.0	517	1067	0,004	10
Phex	0.6	393	1078	0,007	8
	2.0.0	897	3215	0,004	18
	2.8.0	1205	4352	0,003	16
	3.0.0	1419	6036	0,003	19
	3.4.2	1352	5480	0,003	20
Squirrel-sql	1.0	424	717	0,004	15
	2.0	729	1592	0,003	13
	2.6	940	1765	0,002	14
	3.0	1134	2570	0,002	16
	3.1	1223	2989	0,002	16
JSch	0.1.1.4	80	202	0,032	4
	0.1.20	83	191	0,028	5
	0.1.26	94	210	0,024	5
	0.1.34	109	271	0,023	5
	10.1.42	117	385	0,02	5
Hibernate	3.0	956	2739	0,003	19
	3.1	1118	3746	0,003	20
	3.2	1302	4102	0,002	23
	3.3.0	1690	5707	0,002	21
	3.5.1	2446	5980	0,001	21

A. *Número de classes*

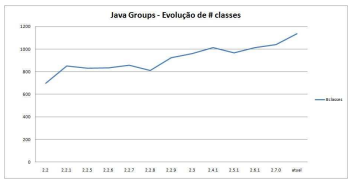
O estudo realizado mostra que o número de classes de um software cresce continuamente. A Figura 1 mostra a evolução do número de classes para os quatro programas. Em 50% dos casos, as versões finais têm mais do que o dobro de classes das versões iniciais. O crescimento drástico do tamanho do software aberto pode ser explicado pelo fato de muitos desenvolvedores colaborarem com o desenvolvimento e, então, o software recebe melhorias e novas funcionalidades continuamente.

B. *Densidade*

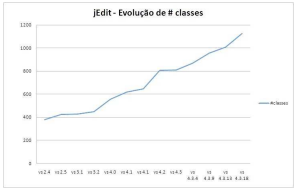
A análise da evolução da densidade da rede mostra que há uma tendência à diminuição de seu valor à medida que o software cresce em número de classes. A Figura 2 mostra a evolução da densidade para quatro dos sistemas de software analisados. A partir dos resultados alcançados pode-se concluir que para cada vértice (classe) inserido no sistema, o número máximo de arestas no gráfico cresce em n unidades, sendo n o número de classes já existentes no software. Porém, o que ocorre é que o número de arestas realmente criadas com a inclusão de uma nova classe é bem menor do que esse valor, o que leva à diminuição da densidade da rede. Em termos de construção de software, isso significa que uma classe nova no sistema tende a se conectar a um número pequeno de outras classes, seja utilizando serviços delas ou prestando serviços a ela.



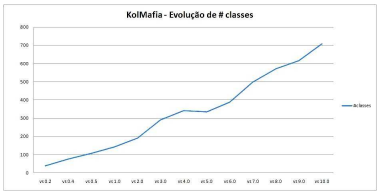
(a)



(b)

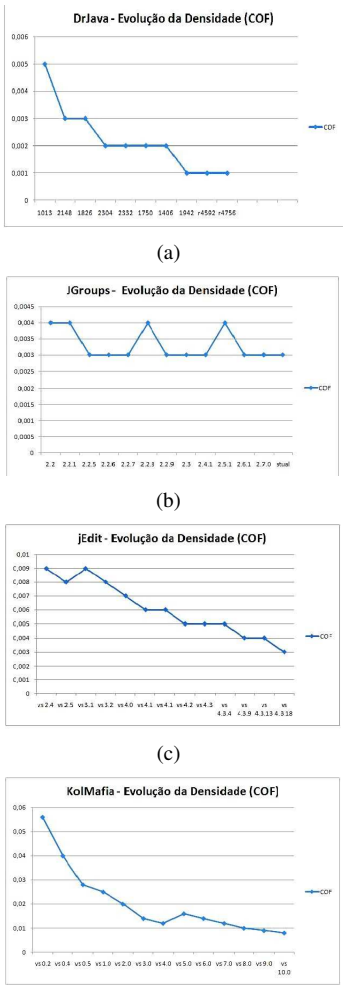


(c)



(d)

Figura 1. Evolução do número de classes

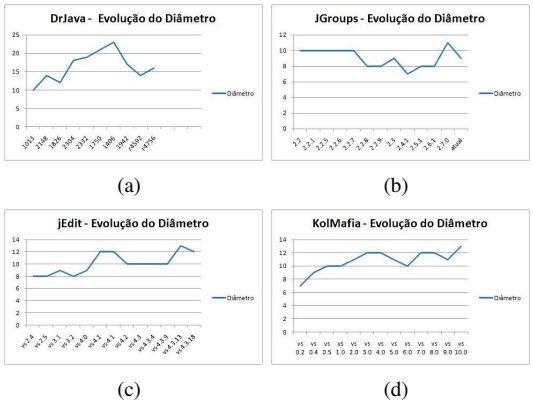


(a) (b) (c) (d)

Figura 2. Evolução da densidade da rede (COF)

C. Diâmetro

O diâmetro das redes analisadas tende a crescer lentamente com o crescimento do software em número de classes. A Figura 2 mostra a evolução do diâmetro para quatro dos sistemas de software analisados. No caso do KolMafia, por exemplo, o número de classes cresceu de 39 para 1109, enquanto o diâmetro cresceu de 7 para 13, sendo que a partir de certo ponto, ele manteve-se praticamente constante, variando de 10 a 12. O diâmetro de uma rede indica o maior caminho mínimo entre os pares de vertices da rede. No caso de software, a existência de um caminho entre uma classe *B* a uma classe *A* indica que uma alteração ou um erro em *A* pode gerar impacto em *B*. Quanto menor esse caminho, maiores as chances de esse impacto chegar a *B*. Como os diâmetros dessas redes tendem a crescer lentamente, significa que o aumento do software aumenta pouco o impacto de alteração de uma classes nas outras em relação à versão anterior.



(a) (b) (c) (d)

Figura 3. Evolução do diâmetro da rede

D. Evolução de classes com alto grau de entrada

A investigação relatada nesta seção busca responder a seguinte pergunta: *as classes mais conectadas de um software tendem a manter essa condição ao longo do crescimento do software?* Os resultados obtidos neste estudo indicam que sim. As Figuras 4, 5 e 6 mostram as classes com maior grau de entrada em três versões de cada um dos sistemas de software JEdit, Java Groups e KoLMafia, respectivamente. São mostrados os dados da versão final, de uma versão intermediária e de uma versão recente. Observa-se que as classes que têm alto grau de entrada na primeira versão apresentam essa característica também nas versões subsequentes. Esse fato também foi observado para os demais sistemas analisados neste estudo. Isso leva a concluir que quando uma classe é incluída no sistema, ela utiliza preferencialmente classes que já são muito utilizadas no sistema. Isso é uma explicação para a distribuição de grau de entrada seguir *power law* e as redes que representam software serem, então, *scale-free*.

Jedit Versão 3.1

Classe	#Conexões
org.gjt.sp.jedit.jEdit	117
org.gjt.sp.util.Log	55
org.gjt.sp.jedit.Buffer	52
org.gjt.sp.jedit.View	49
org.gjt.sp.jedit.GUIUtilities	47
org.gjt.sp.jedit.MiscUtilities	36
bsh.SimpleNode	36
bsh.Node	35
bsh.Primitive	33
org.gjt.sp.jedit.textarea.JEditTextArea	32

(a)

Jedit Versão 4.1.8

Classe	#Conexões
org.gjt.sp.jedit.jEdit	168
org.gjt.sp.jedit.GUIUtilities	85
org.gjt.sp.jedit.View	75
org.gjt.sp.util.Log	74
org.gjt.sp.jedit.Buffer	72
org.gjt.sp.jedit.MiscUtilities	68
org.gjt.sp.jedit.textarea.JEditTextArea	46
org.gjt.sp.jedit.io.VFSManager	41
bsh.SimpleNode	40
bsh.Node	37

(b)

Jedit Versão 4.3.18

Classe	#Conexões
org.gjt.sp.jedit.jEdit	282
org.gjt.sp.util.Log	159
org.gjt.sp.jedit.GUIUtilities	135
org.gjt.sp.jedit.View	102
org.gjt.sp.jedit.Buffer	73
org.gjt.sp.jedit.MiscUtilities	62
org.gjt.sp.jedit.io.VFSManager	51
org.gjt.sp.jedit.textarea.JEditTextArea	49
org.gjt.sp.jedit.buffer.JEditBuffer	45
org.gjt.sp.jedit.bsh.SimpleNode	44

(c)

Figura 4. Evolução de classes com maior grau de entrada - JEdit

Jgroups 2.2	
Classe	#Conexões
org.jgroups.log.Trace	192
org.jgroups.Message	149
org.jgroups.util.Util	135
org.jgroups.Event	98
org.jgroups.View	88
org.jgroups.stack.Protocol	83
org.jgroups.JChannel	70
org.jgroups.Channel	56
org.jgroups.Header	42
org.jgroups.stack.IpAddress	35

(a)

JGroups 2.2.8	
Classe	#Conexões
org.jgroups.util.Util	183
org.jgroups.Message	181
org.jgroups.Event	127
org.jgroups.View	101
org.jgroups.stack.Protocol	99
org.jgroups.JChannel	85
org.jgroups.Channel	58
org.jgroups.stack.IpAddress	55
org.jgroups.Header	47
org.jgroups.util.Queue	44

(b)

JGroups 2.8	
Classe	#Conexões
org.jgroups.util.Util	345
org.jgroups.Message	233
org.jgroups.View	148
org.jgroups.JChannel	148
org.jgroups.Event	135
org.jgroups.stack.Protocol	113
org.jgroups.MembershipListener	107
org.jgroups.Channel	105
org.jgroups.MessageListener	95
org.jgroups.Receiver	77

(c)

Figura 5. Evolução de classes com maior grau de entrada - Java Groups

KolMafia 0.2	
Classe	#Conexões
net.sourceforge.kolmafia.KoLFrame	8
net.sourceforge.kolmafia.KoLmafia	7
net.java.dev.spellcast.utilities.ActionVerifyPanel	6
net.sourceforge.kolmafia.KoLRequest	5
net.java.dev.spellcast.utilities.LockableListModel	4
net.sourceforge.kolmafia.KoLFrame$KoLPanel	4
net.sourceforge.kolmafia.AdventureFrame	3
net.java.dev.spellcast.utilities.ActionVerifyPanel$VerifyButtonPanel	3
net.java.dev.spellcast.utilities.JComponentUtilities	3
net.java.dev.spellcast.utilities.ActionVerifyPanel$VerifiableElement	3

(a)

KolMafia 6.0	
Classe	#Conexões
net.sourceforge.kolmafia.KoLConstants	149
net.java.dev.spellcast.utilities.UtilityConstants	133
net.sourceforge.kolmafia.KoLmafia	117
net.sourceforge.kolmafia.KoLFrame	106
net.sourceforge.kolmafia.KoLCharacter	80
net.sourceforge.kolmafia.KoLRequest	79
net.sourceforge.kolmafia.AdventureResult	66
net.java.dev.spellcast.utilities.LockableListModel	59
net.java.dev.spellcast.utilities.ActionPanel	49
net.sourceforge.kolmafia.StaticEntity	40
net.java.dev.spellcast.utilities.JComponentUtilities	39

(b)

KolMafia 12.0	
Classe	#Conexões
net.sourceforge.kolmafia.KoLConstants	298
net.java.dev.spellcast.utilities.UtilityConstants	251
net.sourceforge.kolmafia.StaticEntity	173
net.sourceforge.kolmafia.KoLmafia	161
net.sourceforge.kolmafia.AdventureResult	139
net.sourceforge.kolmafia.KoLSettings	129
net.sourceforge.kolmafia.RequestThread	125
net.java.dev.spellcast.utilities.LockableListModel	124
net.sourceforge.kolmafia.KoLRequest	116
net.sourceforge.kolmafia.KoLCharacter	108
net.sourceforge.kolmafia.KoLFrame	107

(c)

Figura 6. Evolução de classes com maior grau de entrada - KolMafia

E. Instabilidade de Classes de Alto Grau de Entrada

É possível acreditar que uma classe que, na construção inicial do sistema, tenha alto grau de entrada possa se manter estável ao longo da vida do software. Estabilidade, aqui, refere-se à ausência ou pouca ocorrência de modificações na classe. Intuitivamente, poder-se-ia pensar que, se o software foi bem projetado e aplicaram-se corretamente o princípio do aberto-fechado (*open-closed principle*), que define que um módulo deve estar fechado para alterações e aberto para extensão, então classes sofrerão poucas alterações e, idealmente, não serão alteradas. Além disso, como a classe é grande prestadora de serviços, ela deve ter sido bem testada e depurada e, então, as chances de ela sofre modificações, corretivas ou não, são pequenas. Porém, os dados observados neste estudo mostram que isso não ocorre. Ao contrário, classes com alto grau de entrada tendem a ser muito instáveis. A cada nova versão, elas crescem em número de métodos públicos, em número de atributos públicos e têm sua coesão diminuída. As Figuras 7, 8 e 9 mostram os dados de evolução de classes com alto grau de entrada nos programas JEdit, Java Groups e KolMafia, respectivamente. Uma explicação para esse fato é que como as classes são prestadoras de serviços, a prática que geralmente se aplica é manter essas classes como prestadoras de serviços, incluindo novos serviços nelas para que elas possam atender também às novas classes do sistema. Isso gera uma tendência à deterioração da classe, pois sua coesão diminui. Esse resultado mostra que o princípio *open-closed* e as técnicas de refatoração de software talvez não sejam bem aplicadas na prática.

JEdit: org.gjt.sp.jedit.jEdit				
Versão	#Conexões	Coesão	Atributos públicos	Métodos públicos
2.4	124	0,333	0	23
2.5	123	0,333	0	58
3.1	117	0,167	0	69
3.2	122	0,5	0	71
4.0	146	0,5	0	78
4.1.1	160	0,5	0	82
4.1.8	168	0,5	0	81
4.2	220	0,5	0	91
4.3	216	0,25	0	91
4.3.4	226	0,5	0	91
4.3.9	244	0,25	0	91
4.3.13	254	0,143	0	96
4.3.18	282	0,143	0	109

(a)

Figura 7. Instabilidade de Classes com alto grau de entrada - JEdit

JavaGroups: org.jgroups.Message				
Versão	#Conexões	Coesão	Atributos públicos	Métodos públicos
2.2	149	1	0	25
2.2.1	182	1	0	31
2.2.5	182	0,5	0	31
2.2.6	182	0,5	0	31
2.2.7	186	0,5	0	31
2.2.8	181	0,333	0	33
2.2.9	185	0,5	0	34
2.3	198	0,5	0	35
2.4.1	202	0,5	0	35
2.5.1	196	1	3	39
2.6.1	207	1	3	39
2.7.0	224	0,5	3	39
2.8	233	0,5	6	44

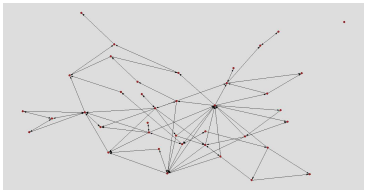
(a)

Figura 8. Instabilidade de Classes com alto grau de entrada - Java Groups

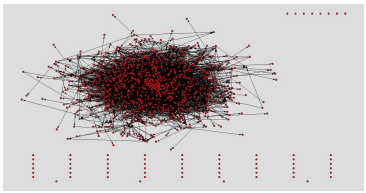
KolMafia: net.sourceforge.kolmafia.KolMafia				
Versão	#Conexões	Coesão	Atributos públicos	Métodos públicos
vs 0.2	7	0,5	0	18
vs 0.4	23	0,5	0	18
vs 0.5	33	0,333	0	23
vs 1.0	55	0,5	0	28
vs 2.0	69	0,333	0	30
vs 3.0	110	0,25	0	48
vs 4.0	134	0,25	0	55
vs 5.0	142	0,143	0	74
vs 6.0	117	0,167	0	63
vs 7.0	150	0,2	1	82
vs 8.0	153	0,167	6	94
vs 9.0	154	0,333	6	77
vs 10.0	205	0,167	19	84
vs 11.0	145	0,067	8	85
vs 12.0	161	0,059	11	88
vs 13.0	218	0,045	11	87
vs 13.7	264	0,05	17	78

(a)

Figura 9. Instabilidade de Classes com alto grau de entrada - Kolmafia

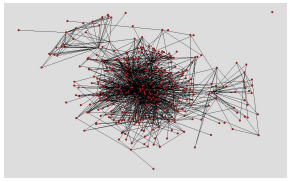


(a)

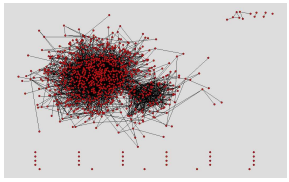


(b)

Figura 10. Visualização gráfica da rede - KolMafia



(a)

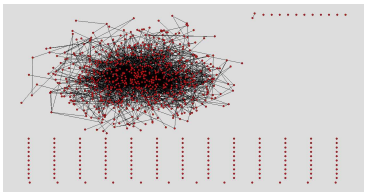


(b)

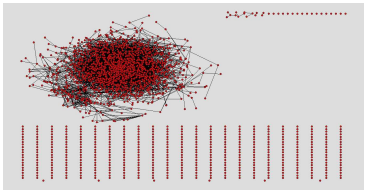
Figura 11. Visualização gráfica da rede - JEdit

F. Análise Gráfica da Redes

A análise gráfica das redes formadas por classes em software aberto orientado por objetos mostra que, embora eles possam ser organizados em pacotes e seguirem diretrizes de organização de software, eles parecem ser constituídos por um grande componente, classes periféricas e, em alguns casos, outros componente pequenos. Neste grande componente, estão a maior parte das classes e há grande interação entre elas. Esse componente envolve cada vez mais classes do software, à medida que ele evolui. As Figuras 10, 11 e 12 mostram a rede formada pelas classes dos programas KolMafia, JEdit e Hibernate, respectivamente, em suas versões iniciais e nas mais recentes.



(a)



(b)

Figura 12. Visualização gráfica da rede - JEdit

V. Trabalhos Relacionados

A observação empírica de redes reais levaram à definição de modelos que fornecem compreensão e predição de comportamento de tais redes. O trabalho de Newman [5] apresenta uma abrangente revisão sobre a evolução dos estudos nesta área, denominada Redes Complexas. Assim como muitas outras redes, também as redes formadas por módulos de um software são objeto de estudo desta área. Têm-se se observado, por exemplo, que algumas características de software, como o grau de entrada de seus módulos seguem *power law* [10], [11], [12], [13], [14], [15]. Uma *power law* é uma função de distribuição de probabilidades na qual a probabilidade de uma variável randômica X assumir um valor x é proporcional a uma potência negativa de x. Essa relação é mostrada na Equação 1.

$$P(X = x) \propto cx^{-k} \tag{1}$$

O fato de o grau de entrada de módulos seguirem leis de potência indica que há um percentual pequeno de classes com alto grau de entrada e um percentual muito grande classes com baixo grau de entrada. As redes formadas pelos módulos em softwares são, então, *scale-free*. Esta informação é muito útil pois permite concluir, dentre outros aspectos, que há um número reduzido de classes no software cujas modificações têm grande impacto nas demais classes, porém, quando uma manutenção ou um erro ocorre nessas classes, os efeitos no software podem ser danosos.

Wheeldon e Counsell [14] analisaram três sistemas de software bem conhecidos, JDK (*Java Development Kit*), Apache Ant e Tomcat, e identificaram *power laws* na distribuição dos graus de entrada das classes. Louridas et al. [11] analisaram 11 sistemas de software desenvolvidos em C, Perl, Java e Ruby, e concluiram que os graus de entrada e saída são regidos por *power law*. Em um estudo que envolveu 40 softwares abertos orientados por objetos desenvolvidos em Java, Ferreira et al. [15] também identificaram *power law* nos graus de entrada das classes. Baxter et al. [10]

e Ferreira et al. [15] realizaram estudos de caracterização de software aberto Java a partir de métricas de software, sendo que Ferreira et al. [15], a partir da análise realizada, identificam os valores mais comuns das métricas analisadas em software aberto Java. O presente trabalho realiza uma caracterização de software aberto, porém do ponto de vista de sua evolução. Busca-se identificar como esses softwares evoluem, a partir da observação de como suas métricas, de software e de rede, evoluem.

Muitos trabalhos têm sido realizados a respeito de evolução de software. Uns dos mais conhecidos são os Lehman [1], que resultaram nas *Leis de Lehman* que postulam, dentre outros aspectos que: um software sofre modificações contínuas para se adaptar ao ambiente; a complexidade do software tende a aumentar; o conteúdo funcional de um programa é continuamente aumentado para manter a satisfação do usuário; programas apresentam qualidade descrescente

Godfrey et al. [2] realizaram um estudo de caso sobre a evolução do núcleo do Linux. O estudo conclui que o crescimento do sistema é contínuo e se dá em taxa geométrica. Wang et al. [3] propõem um conjunto de métricas para avaliar evolução de software aberto. As métricas avaliam aspectos relacionados aos módulos constituintes do sistema, a erros identificados ao longo de sua vida e a implementação de novos requisitos, tais como quantidade de módulos no sistema em um dado momento, número de desenvolvedores que contribuem com o sistema em um determinado momento, número de erros no sistema, número de erros corrigidos e número de requisitos incluídos no sistema. Os autores relatam os resultados de um estudo de caso sobre a evolução do Ubuntu, um distribuição do Linux. Eles observam que a evolução do Ubuntu se dá por inclusão de novos pacotes. Desta forma, nesse estudo de caso pacotes são considerados os módulos do software. Foram consideradas no estudo 5 versões do sistema, entre 2004 e 2006. Os resultados do estudo mostram que o número de módulos do sistema passou de 269 a 1658 ao longo dos dois anos e que o número desenvolvedores que contribuem com o sistema passou de 13 para 280. Com os resultados do estudo, os autores concluem que o crescimento do número de módulos é coerente com o crescimento do número de desenvolvedores e que o crescimento do número de erros identificados é coerente com o crescimento do número de usuários ativos do sistema.

Leskovec et al. [6] descrevem os resultados de um estudo que realizaram com um grande conjunto de grafos reais. Nas redes estudadas eles concluiram que a maior parte dos grafos tornam-se mais densos ao longo do tempo, o que significa que o número de arestas crescem superlineramente com o número de nodos. Outro achado importante é que o diâmetro dessas redes diminui ao longo do tempo. O conjunto de dados são de 12 grafos: citações na área de física, citações na área de literatura, citações de patentes americanas, um grafo da Internet, cinco grafos bipartites de autores e artigos escritos por eles, uma rede de recomendações e uma rede de comunicações por e-mail. Os resultados encontrados no estudo relatado neste artigo mostram que a rede que representa software tende a se tornar menos densa e seu diâmetro tende a crescer, constrastando com os achados de [6].

Um estudo recente de Xie et al.[4] avaliou evolução de software aberto, analisando diferentes versões de 7 sistemas de software abertos. O objetivo do estudo é verificar a aplicabilidade das Leis de Lehman em software aberto. o estudo concluiu que as leis de modificação contínua, complexidade crescente, auto-regulação e crescimento contínuo são observadas em software aberto, enquanto as demais leis não foram observadas. Além disso, o estudo concluiu que um grande percentual de modificações ocorrem em um pequeno percentual de código fonte e que as modificações de interface são mais frequentes do que modificações de implementação e que elas tendem a ocorrer nas fases iniciais da vida do software.

VI. Conclusões

Este trabalho realizou um estudo sobre a evolução de software. Foram analisadas ao todo 109 versões de 16 sistemas de software abertos desenvolvidos em Java, desenvolvidos no período de 2001 a 2009. A análise realizadas baseou-se em métricas de software e de rede. A partir dos resultados do estudo, observou-se que a densidade da rede que representa o software tende a diminuir à medida que ele cresce em número de classes. Isso leva a concluir que novas classes no software aumentam o seu tamanho, o que aumenta também o esforço de manutenção do software, porém isso não aumenta a densidade do software (COF), o que contribui para que sua manutenção continue factível. Outra observação importante é que os graus de entrada (conexões aferentes) seguem *power law*, ou seja, a maior parte das classes têm poucas conexões aferentes e poucas classes concentram a maior parte das conexões aferentes. Estas classes têm grande importância no software, pois manutenções nela geram impacto em um grande número de classes.

Classes com alto grau de entrada (conexões aferentes) tendem a manter essa situação ao longo da vida do software. Essa observação evidencia que classes novas em um sistema usam serviços preferencialmente de classes que já possuem um grande número de classes usuárias. Observou-se também que a conjectura de que classes com muitas conexões aferentes são estáveis é falha. Ao contrário, as classes que possuem o maior número de conexões aferentes tendem a continuar nesta condição durante toda a vida do sotware, e tendem a aumentarem consideravelmente de tamanho e a terem coesão interna piorada. Uma explicação possível para esse fenômeno é que essas classes nascem como grandes fornecedoras de serviços e, à medida que o software cresce,

novos serviços são enxertados nela a fim de atender as novas classes. Isso resulta, possivelmente, da não refatoração do software em classes mais coesas.

A análise gráfica dos grafos referentes aos programas mostra que, mesmo sendo divididos em pacotes, eles parecem ser constituídos por um componente central, que concentra a maior parte das conexões, e alguns nodos e componentes periféricos.

Os resultados do estudo fornecem novas informações relevantes a respeito da estrutura e da evolução de software aberto. É importante estender o estudo para que seja realizada uma análise por tipos de software, como *frameworks*, bibliotecas e aplicações, a fim de se observar se há comportamento diferente relevante na evolução desses tipos de software. É importante também realizar um estudo semelhante como software proprietário a fim de se verificar em que aspectos a evolução desse tipo de software se diferencia da evolução de software aberto.

AGRADECIMENTOS

Os autores agradecem à Fapemig, entidade financiadora do projeto CONNECTA - Conectividade em Módulos (Processo: CEX APQ-3999-5.01/07, Fapemig - Edital Universal), no qual foi realizado o trabalho apresentado neste artigo.

REFERÊNCIAS

[1] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution - the nineties view," in *Proc. of the Fourth Intl. Software Metrics Symposium (Metrics'97)*.

[2] M. Godfrey and Q. Tu, "Growth, evolution, and structural change in open source software," in *Proc. of the IWPSE*.

[3] Y. Wang, D. Guo, and H. Shi, "Measuring the evolution of open source software systems with their communities," vol. 32, no. 6, 2007.

[4] G. Xie, J. Chen, and I. Neamtiu, "Towards a better undertanding of software evolution: An empirical study on open source software," in *ICSM*.

[5] M. E. J. Newman, "The structure and function of complex networks," in *SIAM Reviews*, vol. 45, no. 2, 2003, pp. 167–256.

[6] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," vol. 1, no. 1, Março 2007.

[7] R. Abreu, Fernando Brito; Carapuça, "Object-oriented software engineering: Measuring and controlling the development process," in *Proceedings of 4th Int. Conf. of Software Quality*, McLean, VA, USA, Outubro 1994.

[8] K. A. M. Ferreira, *Um Modelo de Predição de Amplitude da Propagação de Modificações Contratuais em Software Orientado por Objetos*. Belo Horizonte, Minas Gerais: Projeto de Tese de Doutorado. DCC-UFMG, 2009.

[9] PAJEK, *Networks / Pajek Program for Large Network Analysis - for Windows*, Acesso em Setembro de 2009. [Online]. Available: http://vlado.fmf.uni-lj.si/pub/networks/pajek/

[10] G. Baxter, M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero, "Undertanding the shape of java software," in *OOPSLA'06*, Oregon, Portland, USA, Outubro 2006.

[11] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," vol. 18, no. 1, Setembro 2008.

[12] A. Potantin, J. Noble, M. Frean, and R. Biddle, "Scale-free geometry in oo programas," vol. 48, no. 5, pp. 99–103, Maio 2005.

[13] D. Puppin and F. Silvestrini, "The social network of java classes," in *SAC'06*, Dijon, França, 2006, pp. 1409–1413.

[14] R. Wheeldon and S. Counsell, "Power law distributions in class relationships." in *Proceedings of 3rd International Workshop on Source Code Analysis and Manipulation (SCAM)*, Setembro 2003.

[15] K. A. M. Ferreira, M. A. S. Bigonha, R. da S. Bigonha, H. Corrêa, and L. F. de O. Mendes, "Valores referência de métricas de software orientado por objetos," in *Simpósio Brasileiro de Engenharia de Software*.