The challenge of software complexity

Kevin Moore, Michel Wermelinger

Computing Department, The Open University, UK

ou@kevin.moore.name, m.a.wermelinger@open.ac.uk

Abstract

Given the interdisciplinary nature of complex network studies, there is a practical need for dialogue between theorists proposing graph measurements and those seeking to apply them into a domain. We consider this in the domain of software complexity by highlighting the distinctive nature of networks representing software's internal structure and also by describing the application of one such proposal, the offdiagonal complexity, against two examples of software. The results showed the promise of using complex networks to measure software complexity but also demonstrated the confounding effects of size. Based on that application we make proposals to improve the dialogue between theory and experiment.

Software's importance and the need to measure complexity

Today's society is heavily dependant on software. It runs our computers, our phones and the internet, while managing economies and communications. This pervasiveness means that any improvement in understanding software has a potentially enormous payback from better project management, control of costs and increased quality. Past practice of software development could be seen as a chimera of art-form and engineering with success or failure in projects seemingly dependent on anecdotal wisdoms. While a comprehensive theoretical framework seems elusive, current and future practice has become increasingly evidence-based and draws from a wide range of disciplines such as psychology, sociology, data-mining and complexity theories.

That software is complex is also largely self-evident. Brooks (1987) (of “The Mythical Man-Month” fame) argues that complexity is one of the fundamental essences associated with software. As such, understanding this inherent property would make great inroads into understanding software overall.

While there are several viewpoints into software such as its cognitive, computational, problem or solution complexity (Cardoso et al. 2000), this paper focuses on the structural complexity of the code, arguing that it provides the most direct understanding of the product.

Software as a complex network

The variety of coding languages, styles and paradigms makes processing and quantifying code hard to generalise. One solution is to abstract the code into a network graph, with vertices representing a chosen unit of code and edges representing an arbitrary relationship between those units. By representing the interconnections between collaborating modules, objects, classes, methods, and subroutines with a network graph, software becomes another domain capable of investigation with the interdisciplinary toolset of complex networks.

The basic technique is well established (Myers 2003; Valverde & Solé 2003) and while more recent developments have for instance considered graphing the entire socio-technical system (Bird et al. 2009), obtaining a measurement that represents the complexity of source code's basic structure and that can be connected to software development practice remains desirable.

Software as typical

Software networks appear as typical complex networks exhibiting both small-world behaviour and having a long and fat-tailed degree distribution obeying a power law. If they are constructed as directed graphs, the degree distributions of the inward and outward links differ, with the exponent for incoming edges being less than that of the outgoing and showing a better fit to the power law (Valverde & Solé 2003; Potanin et al. 2005; Concas et al. 2007; Louridas et al. 2008).

Solé & Valverde (2004) identify software networks as heterogeneous, scale-free and with some modular structure – a characterisation that also includes a wide range of biological and technical systems. Based on an earlier work (Valverde et al. 2002), they suggest this commonality is due to such systems being shaped through a processes of optimisation – a suggestion that reflects software development well.

Technical and biological networks are typically disassortative, i.e. vertices with a high degree preferentially attach to those with low degree, as opposed to social networks which typically show assortative mixing (Newman 2002). Perhaps

unsurprisingly, software networks have been empirically confirmed as disassortative (Solé & Valverde 2004; Gao et al. 2010).

Software networks can therefore be recognised as typical examples of complex graphs, but some aspects of software create distinctive challenges and opportunities.

Software as atypical

Software networks demonstrate a wide variation of size, reflecting the range of available software from small tools to major applications, but are generally large in comparison with other networks commonly used in complexity research (Louridas et al. 2008; Moore 2011; Newman n.d.)

Network	Nodes
Les Miserables character co-appearance	77
American football games	115
Tomcat 4.1.40 (package to package dependencies)	181
C. elegans neural net	302
Netbeans 6.8 (package to package dependencies)	**1,532**
S. cerevisiae protein-protein interaction	1,870
Tomcat 4.1.40 (class to class dependencies)	**2699**
Netbeans 6.8 (class to class dependencies)	**14,378**
AS internet topology	22,963
BEA Weblogic 8.1 middleware platform (classes)	**80,095**

Table 1: Example sizes of real-world networks, with software networks in bold

The same software network can be considered at different resolutions, i.e. by considering different code units as vertices. For example, in code written in Java, a popular programming language, one can consider classes (which group related functions) and packages (which group related classes). While any scale-free network could be considered in the same way, in software these two 'granularity levels' (or equivalent ones for other programming languages) are particularly significant and represent meaningful and deliberate constructs to software developers. It is possible that the complexity of software networks behaves differently at different resolutions while remaining the same coherent network.

Software networks evolve as the code is modified in response to fault fixing and feature requests, but also as a result of refactoring activity. This activity occurs when developers attempt to rework the code structure while preserving functionality. While refactoring is tricky to isolate from other coding activity, this offers a network that has been changed, hopefully simplified, and yet remains functionally the same. Software networks can also evolve by widespread deletion, as functionality is split out of the main product in a sort of software 'cell division'. The reverse can also happen as existing external products are absorbed wholesale. Even under more routine development it is uncertain what growth models are being applied; as a designed product it is clearly neither stochastic nor perfectly deterministic. The earlier suggestion that an optimisation process is at work seems likely, but it is unclear exactly what developers are optimising for.

Despite this apparent chaos, the evolution of software size is well described with an inverse square model that results in a decaying growth curve. In this model S_t is the size value of release t and E is a model parameter (Turski 2006):

$$S_t = S_{t-1} + E/(S_{t-1})^2$$

The evolution of software complexity is not as well described, although it is argued that complexity will increase as software evolves (Lehman & Fernández-Ramil 2006). Directly measuring software complexity by measuring its representation as a complex network firstly requires identifying a proposed measure and then applying it to example software.

Example: Offdiagonal complexity

Proposed by J.C. Claussen (2007) following earlier discussions and preprints, this measure is capable of distinguishing complex networks from those with a regular or random structure. Its basis is the observation that for complex networks the values in a node-node degree correlation matrix are more evenly spread along the offdiagonals. Such correlations between the degrees of pairs of nodes allows the construction of an approximative complexity estimator from the

entropy of the normalised distribution.

We computed the offdiagonal complexity (OdC) of two medium-sized software networks through their evolution (Moore 2011). This required the development of software implementing OdC, a process that encountered practical difficulties such as interpreting the mathematical notations, which appeared to vary between the original and citing authors, limited examples and apparent errors in the examples given. While these issues were neither insurmountable nor unexpected they did cause uncertainty in validating the software implementation.

Two major free and open source software projects, the integrated development environment Netbeans (n.d.) and Apache webserver component Tomcat (n.d.), were used as datasets. The available stable releases of each software project were converted into network graphs and their OdC values taken alongside established size measures, such as the number of Java classes, using a custom toolset christened netMetric (n.d.). For each release two network graphs were created, giving views of the software at different resolutions: one to represent the dependencies between Java packages (referred to as 'p2p') and another to represent dependencies between Java classes ('c2c' and considered the more detailed).

Netbeans showed nearly a fourfold increase in size, supporting previous understandings of software evolution such as Lehman's 6th law of continuing growth (Lehman & Fernández-Ramil 2006). However the evolution of OdC behaved differently, challenging Lehman's 2nd law of increasing complexity.

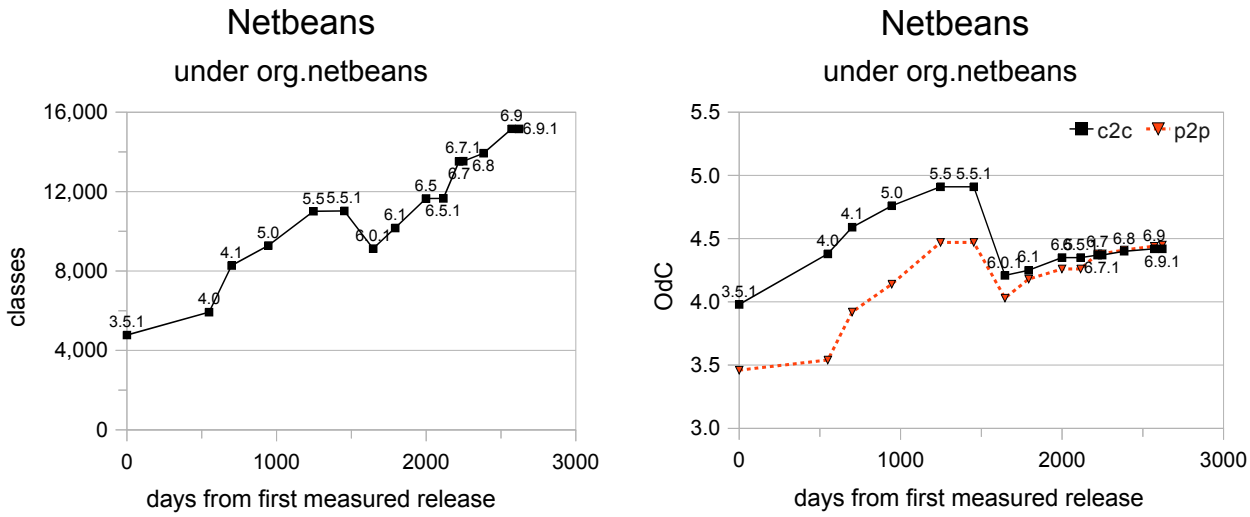

Figure 1: Netbeans size and OdC evolution for packages named org.netbeans. and their classes*

The change of OdC behaviour after release 5.5.1 appears to be due to the removal of J2EE (Java Enterprise Edition) functionality into a separate product and suggests that removal allowed the product to continue growing in size significantly without comparable OdC increases. While the releases studied for Netbeans were the major stable versions (and not for instance the developers' in-progress snapshots), these releases can be categorised as 'new' or 'maintenance'. As can be seen in Figure 1, there is no discernible difference between new releases and their corresponding maintenance releases (e.g. 5.5 and 5.5.1). Normally, maintenance releases correct defects of the previous release by changing the code within code units instead of changing the software's higher-level structure.

A similar pattern of 'punctuated equilibrium', in which sharp changes are followed by a stable period, has been observed in the evolution of other systems, e.g. in Eclipse (a similar product to Netbeans) (Wermelinger et al. 2011). The most drastic change was observed when the Rich Client Platform was added, causing a major restructuring of Eclipse's software architecture.

Tomcat showed far less distinctive evolution in either size or OdC. This is understandable as a consequence of Tomcat implementing a fixed specification meaning that beyond defect fixes the software changes little.

As well as measuring the entire software system, selected subsystems were investigated in the same manner. In Netbeans, each subsystem demonstrated its own evolutionary pattern for both size and OdC in agreement with other works showing that software evolution proceeds differently in different areas of the codebase (Gall et al. 1997; Godfrey & Tu 2000). Tomcat again showed little evolution within subsystems. These observations on software networks suggest that growth and perhaps complexity arise from localised changes in the network.

Offdiagonal complexity was shown to be realistically computable and to show informative behaviour as the software evolved through its releases. However a strong correlation with size (Pearson's r=0.86) limits its usefulness in evaluating software complexity since size is easier and quicker to measure. However, with refinement, the use of degree correlations in an entropy measure could still provide a measurement distinct from size. Claussen (2008) offers the “full OdC” as a way of comparing networks of different size, and Anastasiadis et al. (2005) replaced the Boltzman-Gibbs

entropy in OdC with the generalised Tsallis, and suggested that changing the parameter involved in Tsallis' entropy could make OdC sensitive to particular structures. Unfortunately there was no suggestion as to what those structures might be. Building the correlation matrix using the idea of a *remaining* degree distribution à la Newman (2002) might also improve sensitivity to structural complexity.

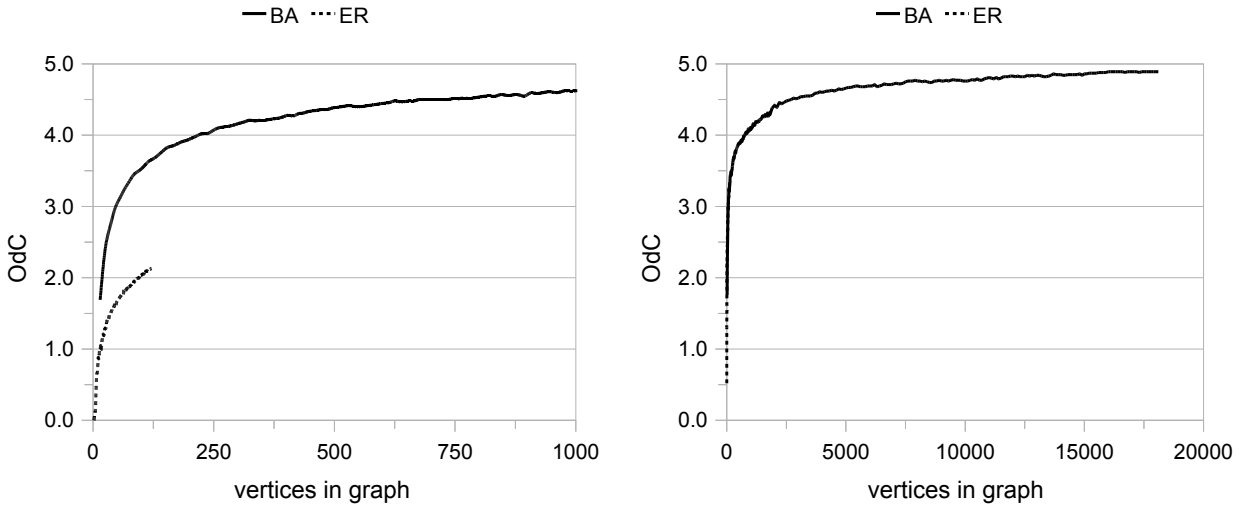

Figure 2: OdC in synthetic networks

We also computed the OdC on simulated Barabási–Albert (BA) and Erdős–Rényi (ER) graphs, observing a rapidly decreasing sensitivity as the number of vertices increased. This suggests that the OdC is most useful for smaller graphs with less than ~300 vertices. These scaling properties demonstrate that measures that appear promising when applied to graphs with tens of vertices lose their practicality applied to the typically much larger software networks. Indeed it suggests that the measure is reflecting a complexity arising from size and not just from structure. This confounding effect of size when measuring complexity is a significant practical issue.

Practical issues

Based on the experience with OdC we make several suggestions for proposed graph measurements that would be helpful for experimentalists, e.g. software engineering researchers like us, interested in complexity metrics.

> The scaling properties should be described. Ideally a proposal should be insensitive to size, but a linear or monotonic relationship with size would still be of practical use since software size can be measured and thus accounted for.
>
> Describing the computability of the metric with a 'big O' notation would allow an assessment of practicality. The availability of this was instrumental in choosing to experiment with OdC.
>
> Providing a reference algorithm in any coding language, including pseudo-code, could improve understanding, especially for non-mathematicians.
>
> Offering downloadable example networks with correct values published would help in verifying software implementations.
>
> A discussion on how the proposal behaves (if at all) against network properties such as diameter or average degree, and what type of network it is relevant for, would help in assessing its suitability to measure software networks. A proposal that for instance focused on polytrees would be unsuitable since they don't represent software networks.
>
> Any suggestions as to what structural features it may be sensitive to would also support the assessment of usefulness.

Ideally this information could be curated into a repository allowing the easy selection of proposals for experiment. While admittedly creating more work for the theorists, the advantage is the increased visibility of their proposal with a faster take up and feedback against real world networks. The nature and form of that feedback should be suggested by theorists as part of establishing a dialogue between theorists and those wanting to apply measurement proposals.

The availability of multiple datasets such as the Qualitas Corpus (Tempero et al. 2010), Helix (Rajesh Vasa & Jones 2010) and the Software-artifact Infrastructure Repository (Do et al. 2005), alongside toolsets for creating call graphs such as netMetric (n.d.), DependencyFinder (n.d.) and Doxygen (n.d.), provide a ready and extensive source of graphs for analysis. Software is a dynamic process with large amounts of ancillary information (such as changelogs) creating software networks whose evolution is potentially observable step-by-step. Measuring complexity in the structure of software remains elusive, but approached through complex networks it is a potentially rich field for study.

Conclusions

In this paper we have shown how software networks offer some distinctive challenges and opportunities when measuring complexity which could be of interest to theorists, particularly in terms of how complex networks evolve. The application of the offdiagonal complexity to a software network has been described and shown to be of interest but limited practical use for measuring software complexity. Based on that, proposals are made in the anticipation of fostering a positive dialogue between theorists proposing graph measures and those investigating their practical application.

Acknowledgements

We thank Jim Hague, from the Physics Department, and Jozef Siran, from the Mathematics Department, for comments on a draft of this paper.

References

Anastasiadis, A.; Costa, L.; Gonzáles, C.; Honey, C.; Széliga, M. & Terhesiu, D. (2005) "Measures of Structural Complexity in Networks", *Complex Systems Summer School 2005, Santa Fe*.

Bird, C.; Nagappan, N.; Gall, H.; Murphy, B. & Devanbu, P. (2009) "Putting it all together: Using socio-technical networks to predict failures", *ISSRE'09. the 20th International Symposium on Software Reliability Engineering*, pp.109-119.

Brooks, F.P., J. (1987) "No Silver Bullet Essence and Accidents of Software Engineering", *Computer*, vol.20, pp.10-19.

Cardoso, A.; Crespo, R. & Kokol, P. (2000) "Two different views about software complexity", *Escom 2000*, pp.433-438.

Claussen, J. C. (2008) "Offdiagonal Complexity: A Computationally Quick Network Complexity Measure - Application to Protein Networks and Cell Division", *Arxiv preprint arXiv:0712.4216*, vol.V, pp.279-287.

Claussen, J. C. (2007) "Offdiagonal complexity: A computationally quick complexity measure for graphs and networks", *Physica A: Statistical Mechanics and its Applications*, vol.375, pp.365 - 373.

Concas, G.; Marchesi, M.; Pinna, S. & Serra, N. (2007) "Power-laws in a large object-oriented software system", *IEEE Transactions on Software Engineering*, vol.33, pp.687-708.

DependencyFinder (n.d.) *Dependency Finder* [online], http://depfind.sourceforge.net/ [accessed 15-April-2010]

Do, H.; Elbaum, S. & Rothermel, G. (2005) "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact", *Empirical Software Engineering*, vol.10, pp.405-435.

Doxygen (n.d.) *Doxygen* [online], http://www.stack.nl/~dimitri/doxygen/ [accessed 19-March-2012]

Gall, H.; Jazayeri, M.; Klosch, R. & Trausmuth, G. (1997) "Software evolution observations based on product release history", *Proceedings on the International Conference on Software Maintenance*, pp.166-.

Gao, Y.; Xu, G.; Yang, Y.; Liu, J. & Guo, S. (2010) "Disassortativity and degree distribution of software coupling networks in object-oriented software systems", *IEEE International Conference on Progress in Informatics and Computing (PIC),*, pp.1000 -1004.

Godfrey, M. & Tu, Q. (2000) "Evolution in open source software: a case study", *Proceedings of the International Conference on Software Maintenance*, pp.131-142.

Lehman, M. M. & Fernández-Ramil, J. C. (2006). "Rules and Tools for Software Evolution Planning and Management", in: Madhavji N., Fernández-Ramil J., P. D. (Ed.), *Software evolution and feedback : theory and practice*, John Wiley \& Sons, pp.539-563.

Louridas, P.; Spinellis, D. & Vlachos, V. (2008) "Power laws in software", *ACM Transactions on Software Engineering and Methodology*, vol.18, pp.2 (26 pp.).

Moore, K. (2011) "Evaluating offdiagonal complexity as a metric of software evolution", Open University.

Myers, C. (2003) "Software systems as complex networks: structure, function, and evolvability of software collaboration graphs", *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, vol.68, pp.46116-1-15.

Netbeans (n.d.) *Netbeans website* [online], http://netbeans.org/ [accessed 1-Jul-2010]

netMetric (n.d.) *netMetric - a tool for measuring large Java codebases* [online], http://sourceforge.net/p/netmetric [accessed 27-Mar-2012]

Newman, M. (2002) "Assortative mixing in networks", *Physical Review Letters*, vol.89, pp.208701.

Newman, M. (n.d.) *Network data* [online], http://www-personal.umich.edu/~mejn/netdata/ [accessed 1-Mar-2012]

Potanin, A.; Noble, J.; Frean, M. & Biddle, R. (2005) "Scale-free geometry in OO programs", *Communications of the ACM*, vol.48, pp.99-103.

Rajesh Vasa, M. L. & Jones, A. (2010) *Helix - Software Evolution Data Set* [online], http://www.ict.swin.edu.au/research/projects/helix [accessed 19-Mar-2012]

Solé, R. & Valverde, S. (2004) "Information theory of complex networks: On evolution and architectural constraints", *Complex networks*, pp.189-207.

Tempero, E.; Anslow, C.; Dietrich, J.; Han, T.; Li, J.; Lumpe, M.; Melton, H. & Noble, J. (2010) "Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies", *2010 Asia Pacific Software Engineering Conference (APSEC2010)*.

Tomcat (n.d.) *Apache Tomcat website* [online], http://tomcat.apache.org/ [accessed 10-Jul-10]

Turski, W. (2006). "A Simple Model of Software System Evolutionary Growth", in: Madhavji N., Fernández-Ramil J., P. D. (Ed.), *Software evolution and feedback : theory and practice*, John Wiley \& Sons, pp.131-141.

Valverde, S.; Cancho, R. & Sole, R. (2002) "Scale-free networks from optimal design", *EPL (Europhysics Letters)*, vol.60, pp.512.

Valverde, S. & Solé, R. (2003) "Hierarchical Small Worlds in Software Architecture", *Arxiv preprint cond-mat/0307278*.

Wermelinger, M.; Yu, Y.; Lozano, A. & Capiluppi, A. (2011) "Assessing architectural evolution: a case study", *Empirical Software Engineering*, vol.16, pp.623-666.