Logarithmic growth dynamics in software networks

S. Valverde[1] and R. V. Solé[1,2]

[1] *ICREA-Complex Systems Lab, Universitat Pompeu Fabra*
Dr. Aiguader 80, 08003 Barcelona, Spain
[2] *Santa Fe Institute - 1399 Hyde Park Road, Santa Fe, NM 87501, USA*

Abstract. – In a recent paper, Krapivsky and Redner (*Phys. Rev. E*, **71** (2005) 036118) proposed a new growing network model with new nodes being attached to a randomly selected node, as well to all ancestors of the target node. The model leads to a sparse graph with an average degree growing logarithmically with the system size. Here we present compeling evidence for software networks being the result of a similar class of growing dynamics. The predicted pattern of network growth, as well as the stationary in- and out-degree distributions are consistent with the model. Our results confirm the view of large-scale software topology being generated through duplication-rewiring mechanisms. Implications of these findings are outlined.

Introduction. – The structure of many natural and artificial systems can be depicted with networks. Empirical studies on these networks have revealed that many of them display a heterogenous degree distribution $p(k) \approx k^{-\gamma}$, where few nodes (hubs) have a large number of connections while the majority of nodes have one or two links [1]. The existence of hubs has been related to multiplicative effects affecting network evolution [2]. Such topological patterns have been explained by a number of mechanisms, including preferential attachment rules [3] and network models based on simple rules of node duplication [4]. A very simple approach is given by the growing network model with copying (GNC) [5]. The network grows by introducing a single node at a time. This new node links to m randomly selected target node(s) with probability p as well as to all ancestor nodes of each target, with probability q (see fig. 1). The discrete dynamics follows a rate equation [5],

$$L(N + 1) = L(N) + \frac{m}{N} \left\langle \sum_{\mu} (p + q j_{\mu}) \right\rangle,$$ (1)

where L and N are the number of links and nodes, respectively. The second term in the right-hand side describes the copying process, where the average number of links added is given by $p + q j_{\mu}$. The μ index refers to the node μ, to be selected uniformly from among
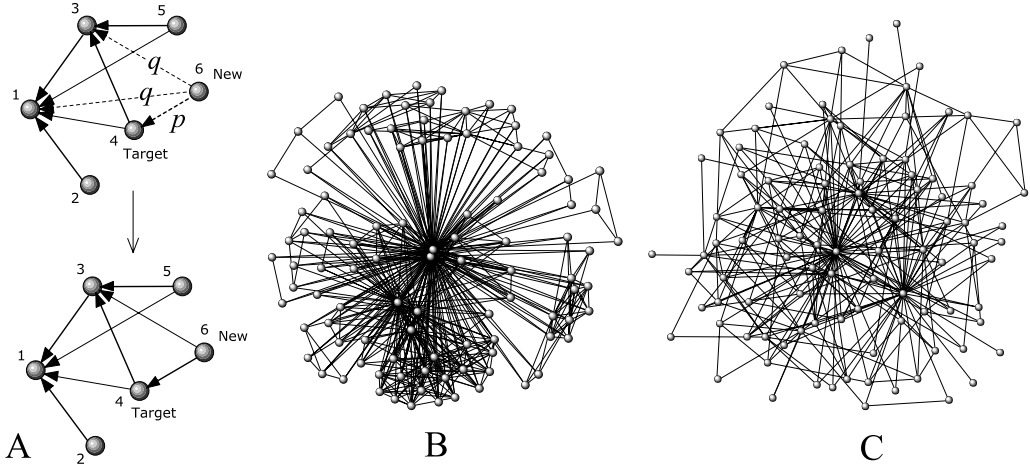
Fig. 1 – (A) Illustration of the copying rule used in the network growth model. Each node is labeled with a number indicating its age (number one is the oldest). In the figure, new node v_6 attaches to target node v_4 with probability p. This new node inherits every link from the target node (dashed links), with probability q. (B) Synthetic network obtained with the GNC model with $N = 100$, $m = 1$, $p = 1$ and $q = 1$. (C) Synthetic network obtained with the GNC model with $N = 100$, $m = 4$, $p = 0.25$ and $q = 0.25$. These networks have a scale-free in-degree distribution and an exponential out-degree distribution.

the N elements. Assuming a continuum approximation, the number of links is driven by the following differential equation:

$$\frac{\mathrm{d}L}{\mathrm{d}N} = mp + mq\frac{L}{N}. \tag{2}$$

The asymptotic growth of the average total number of links depends on the extent of copying defined by the product mq. In particular, logarithmic growth is recovered when $mq = 1$ and $L(N) = mpN \log N$. This corresponds to a marginal situation separating a domain of linear growth ($mq < 1$) to a domain of exponential growth ($2 > mq > 1$). Interestingly, for $mq = 1$ the GNC model predicts a power law in-degree distribution $P_i(k) \approx k^{-\gamma_i}$ with exponent $\gamma_i = 2$ and an exponential out-degree distribution $P_o(k)$, independently of copying parameters. Actually, their derivation for the in-degree distribution can be generalized for arbitrary q and p values, leading to a scaling law $P_i(k) \approx k^{-2}$ for the parameter domain of interest. In ref. [5] the authors showed that the GNC model seems to consistently explain the patterns displayed by citation networks. Here, we show that a GNC model is also consistent with the evolution of software designs, which also display the predicted logarithmic growth.

Software networks. – One of the most important technological networks, together with the Internet and the power grid, is represented by a broad class of software maps. Software actually pervades technological complexity, since the control and stability of transport systems, Internet and the power grid largely rely on sotfware-based systems. In spite of the multiplicity and diversity of objectives and functionalities addressed by software projects, we have pointed out the existence of strong universals in their internal organization [6]. Computer programs are often decomposed in a collection of text files written in a given programming language. In this paper, we will study computer programs written in C or C++ [7]. Program structure can be recovered from the analysis of program files by means of a directed network representation.

In a software network, software entities (files, classes, functions or instructions) map onto nodes and links representing syntactical dependencies [6]. Class graphs (also called "logical dependency graph" [8]) are a particular class of software networks that has been shown to be small-world and scale-free network with an exponent $\gamma \approx 2.5$ [6, 9, 10]. Interestingly, the frequencies of common motifs displayed in class graphs can be predicted by a very simple duplication-based model of network growth [11]. This result indicates that the topology of technological designs, in spite of being planned and problem-dependent, might actually emerge from common, distributed rules of tinkering [12]. In the following, we provide further evidence for the importance of duplication processes in the evolution of software networks.

Here, we study a new class of software networks. We use the so-called "include graph" (or "physical dependency graph" in [8]) $G = (V, E)$, where $v_i \in V$ is a program file and a directed link $(v_i, v_j) \in E$ indicates a (compile-time) dependency between file v_i and v_j. In C and C++, such dependencies are encoded with the keyword "#include" followed by the name of the refereed source file [8]. In order to recover the include graph, we have implemented a network reconstruction algorithm that analyses the contents of all files in the software project looking for this reserved keyword. Every time this keyword is found in a file v_i, the name of the refereed file v_j is decoded and a new link (v_i, v_j) is added. No other information is considered by the network reconstruction algorithm. Notice that the include network is unweighted because it makes no difference to include the same file twice.

In this paper, we investigate the structure and evolution of software maps by looking at their topological structure and the time series of aggregate topological measures, such as number of nodes $N(t)$, number of links $L(t)$ or average degree $k(t) = L(t)/N(t)$. It is worth mentioning that the number of nodes in a include graph coincides with the number of files in the software project, which is often used as a measure of project size.

Software maps typically display asymmetries in their in- and out-degree distributions [9, 10] although the origins of such asymmetry remained unclear. Notice how the out-degree and in-degree distributions of real include networks are quite similar to the corresponding distributions obtained with the GNC model (see previous section). The in-degree and out-degree distributions for the largest component of two different systems (see fig. 2A) are shown in figs. 2B and C, where we have used the cumulative distributions $P_>(k) = \int\limits_k^\infty P(k)\mathrm{d}k$. In both cases, in-degree distributions display scaling $P_i(k) \approx k^{-\gamma_i}$, where the estimated exponent is consistent with the prediction from the GNC model, whereas out-degree distributions are single-scaled (here the average value for the systems analysed is $\langle \gamma_i \rangle = 2.08 \pm 0.04$ [13]). As shown in the next section, these stationary distributions result from a logarithmic growth dynamics consistent with the GNC model.

Software evolution. – Although an extensive literature on software evolution exists (see, for example, [14, 15], little quantitative predictions have been presented so far. Most studies are actually descriptive and untested conjectures about the nature of the constraints acting on the growth of software structures abound. It is not clear, for example, if the large-scale patterns are due to external constraints, path-dependent processes or specific functionalities. In order to answer these questions, we have compared real software evolution with models of network growth, where software size is measured as the number of nodes in the corresponding include graph. In this context, the assumptions of the GNC model are consistent with observations claiming that code cloning is a common practice in software development [15]. Indeed, comparison between real include graphs and those generated with the GNC model suggests the extent of copying performed during software evolution is a key parameter that explains the overall pattern of software growth. Such a situation has been also found in class diagrams [11].
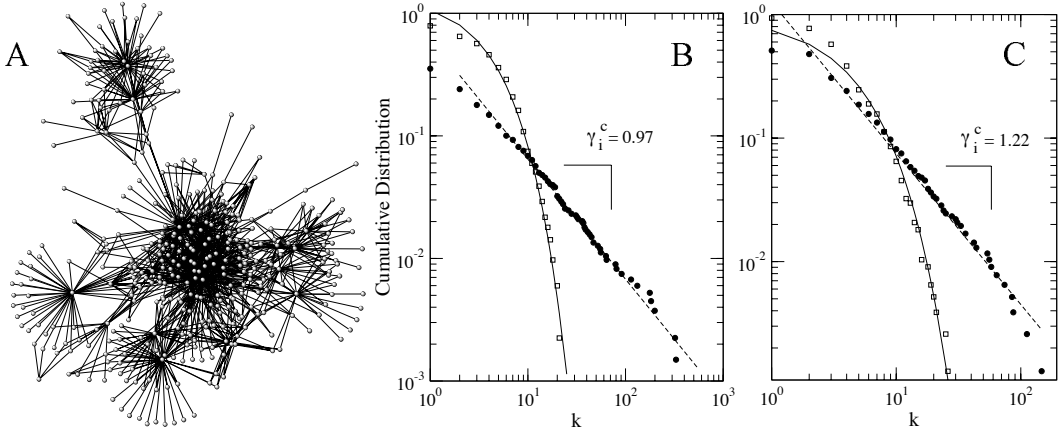
Fig. 2 – (A) The largest connected component of the XFree86 include network at 15/05/1994 (with $N = 393$) displays scale-free behavior (see text). In (B), the cumulative distributions $P_{i>}(k)$ and $P_{o>}(k)$ are shown for a more recent version of the XFree86 include network with $N = 1299$ (not shown here). The power law fit of the in-degree distribution yields $P_i(k) \sim k^{-\gamma_i^c - 1}$ with $\gamma_i^c = 0.97 \pm 0.01$ while the out-degree distribution is exponential. In (C) we can notice similar features for the in-degree and out-degree distributions of the Aztec include network at 29/3/2003. For this system, the power law fit of the in-degree distribution yields an exponent $\gamma_i^c = 1.22 \pm 0.03$.

The growth dynamics found in include graphs is logarithmic (see fig. 3A), thus indicating that we are close to the $mq = 1$ regime. Indeed, the sparseness seen in software maps is likely to result from a compromise between having enough dependencies to provide diversity and complexity (which require more links) and evolvability and flexibility (requiring less connections). Here we have uneven, but detailed information of the process of software building. In this context, different software projects developments display specific patterns of growth. Specifically, the number of nodes N grow with time following a case-dependent functional form $N = \phi(t)$. Using $\mathrm{d}L/\mathrm{d}t = (\mathrm{d}L/\mathrm{d}N)(\mathrm{d}\phi/\mathrm{d}t)$, we have, from eq. (2),

$$\frac{\mathrm{d}L}{\mathrm{d}t} = \left[mp + mq\frac{L}{\Phi(t)} \right] \dot{\Phi}^{-1} \tag{3}$$

with a general solution

$$L(t) = e^{mq \int (\Phi\dot{\Phi})^{-1} \mathrm{d}t} \left[mp \int e^{-mq \int (\Phi\dot{\Phi})^{-1} dt} \dot{\Phi}^{-1} \mathrm{d}t + \Gamma \right], \tag{4}$$

where Γ is a constant. Using a linear law growth (which is not uncommon in software development), *i.e.* $N(t) = N_0 + at$, and assuming $mq = 1$, we have

$$L(t) = (N_0 + at) \left[mp \log \left(\frac{N_0 + at}{N_0} \right) + \frac{L_0}{N_0} \right]. \tag{5}$$

However, typical time series of $L(t)$ in real software evolution is subject to fluctuations (see fig. 3A). In order to reduce the impact of fluctuations, we use the cumulative average degree $K(t) = \int_0^t (L/N)\mathrm{d}t$, instead. Assuming the number of nodes grows linearly in time, we obtain

$$K(t) = \frac{mp(N_0 + at)}{a} \left[\log \left(\frac{N_0 + at}{N_0} \right) - 1 \right] + \frac{L_0}{N_0}t + \frac{mpN_0}{a}. \tag{6}$$
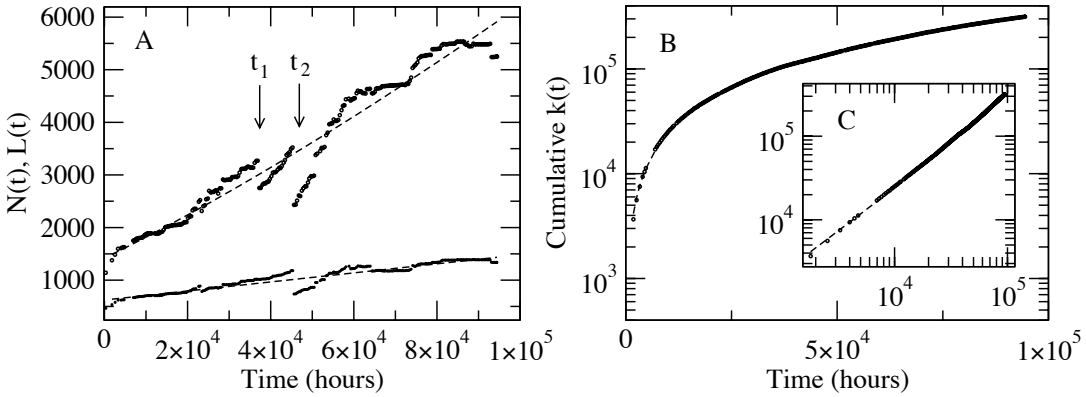
Fig. 3 – (A) The top curve shows the comparison between the time evolution of number of links $L(t)$ in XFree86 between 16/05/1994 and 01/06/2005 (points) and the prediction of eq. (5) (dashed line). In the bottom curve we compare the time evolution of system size $N(t)$ and its linear fitting $N(t) = N_0 + at$ (dashed line). We observe an anomalous growth pattern followed by a discontinuity (here indicated as t_1 and t_2) in $L(t)$. Notice how t_2 signals a discontinuity both in $L(t)$ and $N(t)$, while discontinuity t_1 only takes place in $L(t)$. (B) Comparison between time evolution of the cumulative average degree in XFree86 during the same time period as in (A) and the analytic prediction of eq. (6). (C) The inset shows the same data as in (B) but in a double logarithmic plot. The fitting parameters are: $N_0 = 622.17 \pm 10.92$, $a = 0.0086 \pm 0.0002$, $L_0 = 1419.8 \pm 4.1$, and $mp = 2.20 \pm 0.01$. Time is measured in hours.

The above expressions can be employed to estimate the parameters L_0 and mp describing the shape of the logarithmic growth of number of links $L(t)$ and the parameters N_0 and a controlling the linear growth of the number of nodes $N(t)$. We used the following fitting procedure. For each software project, we have recovered a temporal sequence $\{G_t = (V_t, E_t) \,|\, 0 \leq t \leq T\,\}$ of include networks corresponding to different versions of the software project. Time is measured in elapsed hours since the first observed project version (which can or cannot coincide with the beginning of the project). This temporal sequence describes the evolution of the software project under study. From this sequence, we compute the evolution of the number of nodes $n_0, n_1, n_2, ..., n_T$, the evolution of the number of links $l_0, l_1, l_2, ..., l_T$ and the evolution of the average degree $k_0, k_2, ..., k_i = l_i/n_i, ..., k_T$. In general, available data is a partial set of records of development histories and often misses the initial project versions corresponding to the early evolution. Then, $t_0 \neq 0$ and this explains why the initial observations for n_0 and l_0 are higher than expected. However, we have rescaled time so the first datapoint corresponds to zero. We have collected partial[1] evolution registers for seven different projects (relevant time period is in parenthesis): XFree86 (16/5/94–1/6/05), Postgresql (1/1/95–1/12/04), DCPlus-Plus (1/12/01–15/12/04), TortoiseCVS (15/1/01–1/6/05), Aztec (22/3/01–14/4/03), Emule (6/7/02–26/7/05) and VirtualDub (15/8/00–10/7/05) [13]. The full database comprises 557 include networks (see table I).

Then, we proceed as follows. First, for each software project, its time series for the number of nodes is fitted under the assumption of linear growth, *i.e.* $N(t) = N_0 + at$, and thus yielding N_0 (initial number of nodes) and a (rate of addition of new files). In table I, we can appre-

---

[1]Actually, these datasets constitute a coarse sampling of the underlying process of software change. Collecting software evolution data at the finest level of resolution requires a monitoring system that tracks automatically all changes made by programmers. Instead, it is often the programmer who decides when a software register is created. The issue of fine-grained sampling is an open research question in empirical software engineering that deserves more attention. These limitations preclude us from a more direct testing of the GNC model.

Table I – *Predictions of eq.* (6) *for different systems.*

Project	a	N_0	mp	L_0	T
XFree86	0.0086 ± 0.0001	622.17 ± 10.92	2.20 ± 0.01	1419.80 ± 4.09	243
Postgresql	0.0066 ± 0.0002	601.42 ± 11.35	1.78 ± 0.05	243.89 ± 8.46	31
DCPlusPlus	0.004 ± 0.0001	101.51 ± 2.42	0.70 ± 0.03	338.96 ± 1.30	74
TortoiseCVS	0.0057 ± 0.0001	97.57 ± 2.62	1.59 ± 0.02	105.76 ± 1.58	107
Aztec	0.026 ± 0.002	205.12 ± 22.17	0.97 ± 0.03	622.61 ± 4.77	14
Emule	0.016 ± 0.0006	98.01 ± 6.37	1.65 ± 0.11	223.80 ± 9.34	54
VirtualDub	0.0079 ± 0.0004	167.04 ± 12.44	1.34 ± 0.05	381.50 ± 5.16	35

ciate that the majority of projects grow at a rate a proportional to 10^{-3} files/hour while two medium size projects (Aztec and Emule) actually grow by an order of magnitude faster. Next, we compute the time series of cumulative average degree $K(t)$ by integrating numerically the sequence of k_t values. This new sequence will be fitted with eq. (6) in order to estimate the parameters L_0 (initial number of links) and the product mp controlling the extent of duplication.

In fig. 3B we show the result of the previous fitting procedure to the time series of cumulative average degree $K(t)$ in XFree86, a popular and freely re-distributable open source implementation of the X Windows System [13]. As shown in the figure, the agreement between theory and data is very good. We have validated the same logarithmic growth pattern in the evolution of other software systems (see table I). In particular, we provide a prediction for the average number of links to target nodes, mp, which is found to be small. This is again expected from the sparse graphs that are generated through the growth process.

Together with the overall trends, we also see deviations from the logarithmic growth followed by reset events. In fig. 3A we can appreciate a pattern of discontinuous software growth in the number of links $L(t)$ for XFree86. The time interval delimited by t_1 and t_2 is the signature of a well-known major redesign process that enabled 3D rendering capabilities in XFree86. This new feature of XFree86 was called Direct Rendering Infrastructure (DRI). Development of DRI is cleary visible in the time series of $L(t)$. At t_1 (*i.e.*, August 1998) the design of DRI was officially initiated and the event t_2 (*i.e.*, July 1999) corresponds to the first public release of the DRI technology (*i.e.*, DRI 1.0) [16]. A careful look at the time series $L(t)$ shows that before the discontinuities (indicated by t_1 and t_2), some type of precursor patterns were detectable.

The above example suggests how deviations from the logarithmic growth pattern can predict future episodes of costly internal reorganization (so-called refactorings [17]). In XFree86, the integration of DRI was a costly redesign process characterized by an exponential growth pattern in the number of links $L(t)$. This accelerated growth pattern starts at t_1 and finishes in a clearly visible discontinuity (indicated here by t_2) that signals a heavy removal of links. After t_2 we observe a pattern of fast recovery eventually returning to the logarithmic trend described by eq. (5) (dashed lines in fig. 3A). Such type of reset pattern has been also found in economic fluctuations in the stock market [18]. This trend needs to be explained and might actually result from conflicting constraints leading to some class of marginal equilibrium state. This is actually in agreement with the patterns of activity change displayed by the community of software developers (unpublished results) which also exhibits scale-free fluctuations.

* * *

REFERENCES

[1] DOROGOVTSEV S. N. and MENDES J. F. F., *Evolution of Networks: From Biological Nets to the Internet and WWW* (Oxford University Press, New York) 2003.
[2] BARABÁSI A.-L. and ALBERT R., *Science*, **286** (1999) 509.
[3] CALDARELLI G., CAPOCCI A., DE LOS RIOS P. and MUNOZ M. A., cond-mat/0207366 v2 (2002).
[4] SOLÉ R. V., PASTOR-SATORRAS R., SMITH E. D. and KEPLER T., *Adv. Complex Syst.*, **5** (2002) 43; VAZQUEZ A., FLAMMINI A., MARITAN A. and VESPIGNANI A., *Complexus*, **1** (2003) 38; PASTOR-SATORRAS R., SMITH E. D. and SOLÉ R. V., *J. Theor. Biol.*, **222** (2003) 199; KIM J., KRAPIVSKY P. L., KAHNG B. and REDNER S., *Phys. Rev. E*, **66** (2002) 055101.
[5] KRAPIVSKY P. L. and REDNER S., *Phys. Rev. E*, **71** (2005) 036118.
[6] VALVERDE S., FERRER-CANCHO R. and SOLÉ R. V., *Europhys. Lett.*, **60** (2002) 512.
[7] STROUSTRUP B., *The C++ Programming Language* (Addison Wesley) 1985.
[8] LAKOS J., *Large Scale C++ Software Design* (Addison-Wesley, New York) 1996.
[9] VALVERDE S. and SOLÉ R. V., Santa Fe Inst. Working Paper, SFI/03-07-044 (2003).
[10] MYERS C. R., *Phys. Rev. E*, **68** (2003) 046116.
[11] VALVERDE S. and SOLÉ R. V., *Phys. Rev. E*, **72** (2005) 026107.
[12] SOLÉ R. V., FERRER R., MONTOYA J. M. and VALVERDE S., *Complexity*, **8** (2002) 20.
[13] XFree86 (`http://www.xfree86.org`); Postgresql (`http://www.postgresql.org`); DCPlusPlus (`http://dcplusplus.sourceforge.net`); TortoiseCVS (`http://www.tortoisecvs.org`); Aztec (`http://aztec.sf.net`); Emule (`http://www.emule-project.net`); VirtualDub (`http://www.virtualdub.org`).
[14] BELADY L. A. and LEHMAN M. M., *IBM Systems J.*, **15** (1976) 225.
[15] GODFREY M. and TU Q., *Proceedings of the 2001 International Workshop on Principles of Software Evolution (IWPSE-01), Vienna (2001)*.
[16] HAMMEL M. J., *Linux Magazine*, December (2001).
[17] FOWLER M., BEEK K., BRANT J. and OPDYKE W., *Refactoring* (Addison-Wesley, Boston) 1999.
[18] SORNETTE D. and JOHANSEN A., *Physica A*, **245** (1997) 411.