Project Number 001907

DELIS

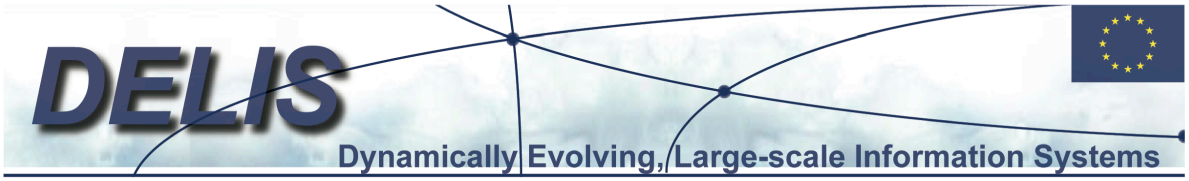
Dynamically Evolving, Large-scale Information Systems

Integrated Project

Member of the FET Proactive Initiative **Complex Systems**

Deliverable D5.2.2

Optimal Strategies for the Collective Construction of Efficient Information-Processing Webs



Start date of the project:	January 2004
Duration:	48 months
Project Coordinator:	Prof. Dr. math. Friedhelm Meyer auf der Heide Heinz Nixdorf Institute, University of Paderborn, Germany
Due date of deliverable:	December 2005
Actual submission date:	January 2006
Dissemination level:	PU – public
Work Package 5.2:	Evolved Tinkering and Degeneracy as Engineering Concepts
Participants:	Universtitat Pompeu Fabra (UPF), Barcelona, Spain Universita di Bologna (UniBO), Italy
Authors of deliverable:	Sergi Valverde (svalverde@imim.es) Ricard V. Solé (ricard.sole@upf.edu) David Hales (hales@cs.unibo.it) Ozalp Babaoglu (babaoglu@cs.unibo.it)

Abstract

This report comprises the complete D5.2.2 deliverable as specified for workpackage WP5.2 in Subproject SP5 of the DELIS (Dynamically Evolving Large-scale Information Systems) Integrated Project.

The essential goal of the DELIS project is to understand, predict, engineer and control large evolving information systems. Increasingly the software development process is being viewed as an interaction between several kinds of evolving network. Program code itself can be graphed as a complex network evolution process at several levels of scale. Also, the programmers working on code are embedded in social networks of developers centred around various projects. Within Open Source development most of the social network evolution process is mediated via electronic support systems rather than direct social contact which allows some degree of tracking of the evolution of both networks (code and social). Can models be constructed that capture these processes? In this deliverable we outline initial investigations in this area[1].

[1]Most papers produced within DELIS are available from the DELIS website as DELIS Technical Reports. Where this is the case references are appended with the DELIS Tech Report number in square brackets. This indicates the paper was produced within the DELIS project, not some other project.

Contents

1 Introduction

Modern information systems are not just driven by humans but artificial computers also play an important role in the resulting information dynamics. For instance, the Internet is a medium that allows any pair of users to communicate, i.e. by e-mail. We can witness human-computer communication taking place in the Internet, where the computer is a remote service that sends the requested information back to the user, i.e. www. In this case, the computer is driven by what is known as *software*, or a sequence of stored machine instructions that indicates with precision the computer what action to do next. In spite of the large body of common-sense practices, heuristics and methodologies used when writing a computer program, i.e., *software engineering*, it is relatively unknown what are the actual mechanisms or rules employed by programmers [9]. At the moment, we cannot give more than recommendations and we do not have any quantitative means to identify what the best practices are. This situation urges the computer science and engineering community to find an *"intellectually rigorous, analytical, teachable design process to ensure development of systems we all can live with"* [5].

One of the consequences of this lack of scientific understanding about software design processes is the impossibility to make reliable predictions about future developments. We would like to give precise answers to questions like: what amount of effort is required to build a software system of a given complexity? Or how should the social network of the software development team be structured for optimum productivity? Frequent and undesirable cost overruns and schedule delays claims for a dramatic increase in confidence levels associated with predictions. On the other hand, little can be learnt from bad experiences because we lack a reference model that enables useful comparison and allows us to evaluate the consequences of software change. Is it possible to give a mathematical model predicting the software development process?

One approach to build this predictive framework exploits strong regularities (also called *invariants*) found in the source code of software systems. The existence of these invariants reveals fundamental principles of software design that, as such, enable accurate predictions. We have suggested that strong constraints might be operating on top of artificial design processes [11, 13, 12]. Proving whether these constraints really exist might be helpful when understanding what limitations influence software development. For example, what structural configurations cannot be reached in the apparently huge space of software design? In this context, we have found that computer programs written in C++ and Java display invariants in their global and local structure.

We have analyzed these invariants by studying networks representing software structures, where network nodes and directed links represent software entities (i.e., classes, files, or subroutines) and syntactical dependencies (i.e., inheritance, aggregation or use), respectively. An example of an exogenous invariant is the fractal structure displayed by software systems indicated by a highly heterogeneous distribution of connectivity between software modules [13]. It turns out that the best path towards understanding the origin of software heterogeneity is modeling the design process itself. Heterogeneity links processes of software change and software structure.

2 Affiliation Networks

Although many software architecture models have been proposed in literature, there is still a need for connecting software designs with its development or evolutionary process. Structural complexity might act as the medium connecting individual behavior and other high-level features of software development, like social structure and the emergence of specialization. It has been argued that the interaction between individuals may have some influence in the organization and structuring of the source code. Distributed teams face pressing communication and coordination problems that can be better solved if software is structured according to social organization [4] which, in turn, can be

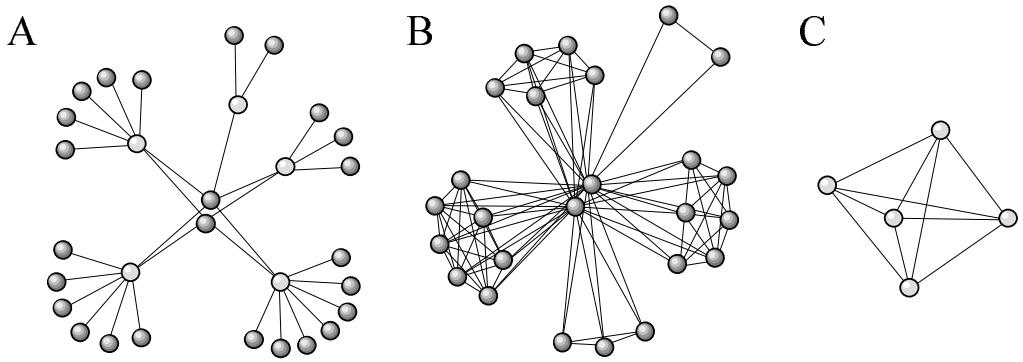


Figure 1: (A) An affiliation network and its two one-mode projections: (B) co-author projection and (C) co-citation network. Dark balls represent authors and light balls represent papers. Both one-mode projections display correlations (i.e. they are highly clustered) even if the association between authors and papers is made at random.

depicted with a social network.

This social dimension of software development can be properly addressed with the affiliation network [15]. An association network is a bipartite graph having two different kinds of nodes. A typical usage of affiliation networks in social sciences describes the simultaneous evolution of author and paper citation networks (see fig. 1). Authors connect articles to one another in co-citation networks, and articles link authors to one another in co-authorship networks. In our context, the two previously mentioned networks (software structure and social developer network) can be integrated in a single and coherent co-evolving framework relating developers with their corresponding software pieces. Here, the affiliation network maps developer tasks (i.e., design, coding or testing) between the social network and the software network. We have built a software system (see fig. 2) that enables us to recover (semi-automatically) and parse CVS log files reporting detailed software development histories (sourceforge.net). A CVS log file provides the information required to recover the affiliation network between developers and project files. Each entry in the CVS log file describes a file revision indicating the modified/created/accessed project file, the name of the developer accessing this file and a timestamp. At this moment, we have collected information about different open-source software projects spanning different interests and developer communities (see table below). As a matter of testing, we have also included a project with only just one developer (DCPlusPlus).

Project	**Revisions**	**Developers**	**Files**
Apache	43698	78	1279
Mozilla	452101	546	28086
FreeBSD	363333	425	28056
OpenBSD	245470	195	33998
Xfree86	27710	21	1788
Inkscape	15423	25	1648
SDCC	9557	32	1318
Gaim	20047	30	767
DCPlusPlus	5260	1	187

Table 1: Some examples of projects analysed. Very different numbers of developpers and activity levels can be observed.

3 Social Simulation of programmer network formation

Recently, some researchers have begun to analyse the social support networks of open source developers which can be traced via weblogs, mailing lists and social networking websites [14]. From such traces it has been possible to produce agent based social simulation models that appear to capture some of the dynamics of project life-cycles.

In such models, essentially, programmers are seen as agents choosing projects in order to satisfy their own individual needs but doing so collectively - such that no single programmer can make an arbitrary change to a project but rather must make changes that are acceptable to most users and developers (who may have different individual needs).

Agents move between projects in order to increase their individual fitness (how well the project reflects their preferences). Projects with no developers left are effectively "dead" and do not progress.

Although at present we are studying this new work we feel that there are interesting parallels with our existing sociologically inspired algorithms producing cooperation in Artificial Social Networks [8].

Specifically, in these of models, which we have worked with extensively, agents move between groups to improve their fitness - with empty groups dying and new groups forming. We have applied such models to firm formation in artificial economies [10] and self-organising peer-to-peer networks [10]. We have also conjectured that a variant of this same group-like selection process could be operating to support the BitTorrent file-sharing protocol [3] in previously not fully comprehended ways [6]. However, in both those cases we did not explicitly model an *external* social network formation process separate from the tasks in hand - it would be interesting to consider this aspect and it's effect on such models in possible future work.

4 Summary

The problem of software development does not reside in technical issues (i.e., specific features of programming languages) but in social aspects of software development. For example, bug location and fixing is an important task in software development and consumes a large fraction of human resources. Bugs are difficult (or even impossible) to remove because the nonlinear character of the phenotype/genotype mapping. In fact, it can be shown that many dynamical properties of computer programs can not be predicted from static information given by the program representation. However, early studies on bug dynamics already suggested a link between software structure and dynamics [1, 2].

In this context, stochastic and generative models of software structure are a prerequisite for quantitative studies on software bug dynamics. We have started to model the simultaneous evolution of the software network and the social network of developers. Only the quantitative characterization of these affiliation networks will enable us to optimize software development processes.

References

[1] Challet, D. and Yann Le Du, (2003) Closed source versus open source in a model of software bug dynamics, preprint cond-mat/0306511.

[2] Challet, D. and Lombardoni, A. (2004) Bug propagation and debugging in asymmetric software structures. *Physical Review E 70, 046109*

[3] Cohen, B. (2003) Incentives Build Robustness in BitTorrent. Presented at the *1st Workshop on the Economics of Peer-2-Peer Systems*, June 5-6, 2003, Berkley, CA. Available at: http://www.sims.berkeley.edu/research/conferences/p2pecon/

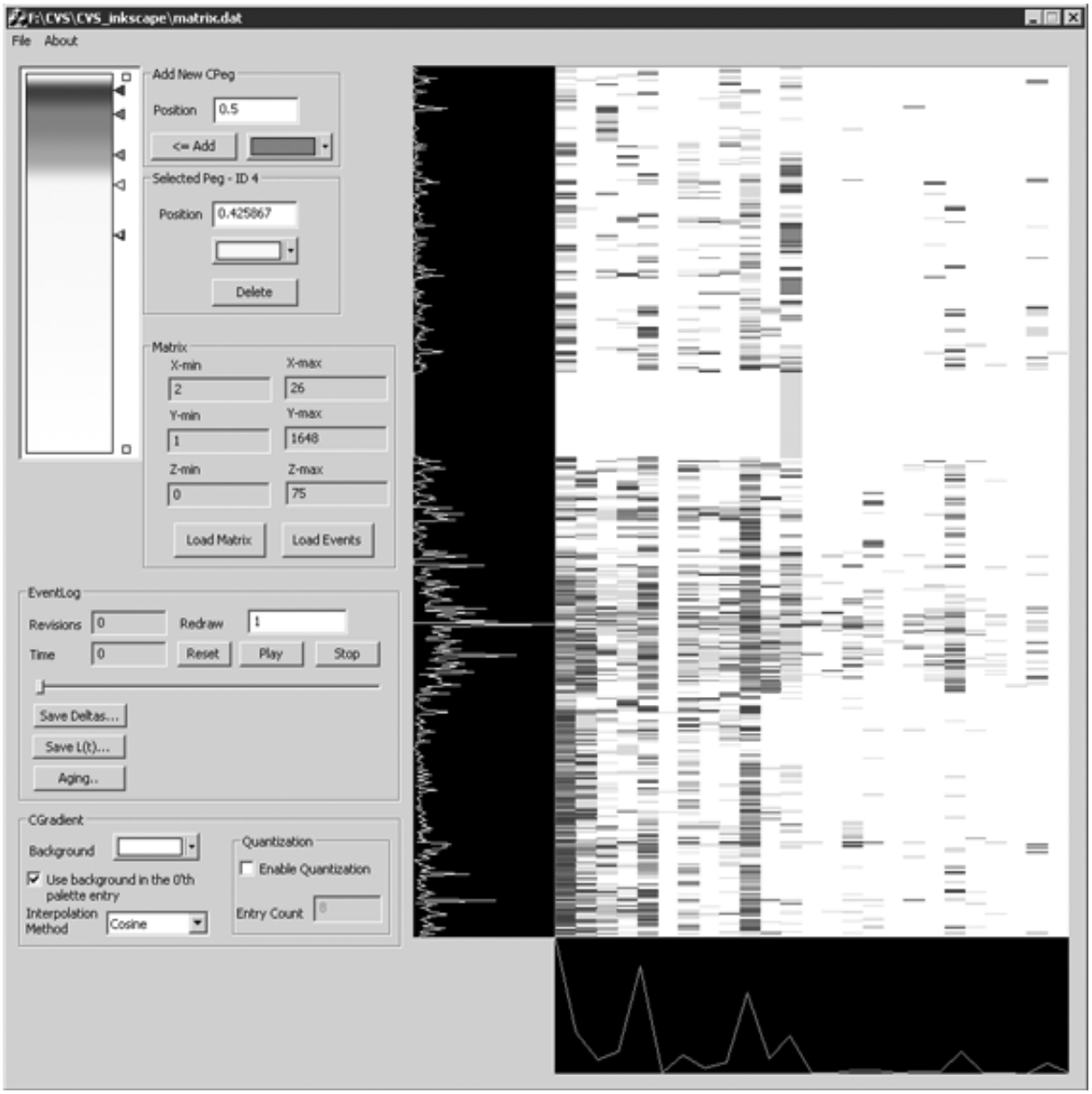


Figure 2: An screenshot from our CVS analysis application. The CVS log file downloaded from the project server is parsed by this application and the affiliation network is recovered. We can see the developer-file affiliation network displayed in the right as an adjacency matrix. Darker color indicates less file accesses (background is white for picture clarity). Columns in this matrix indicate developers in order of appearance (starting from left) while rows indicate project files in order of appearance (starting from bottom). Side plots on the margins of the matrix display the marginal cumulative distributions for files and developers, respectively.

[4] Conway, M. E. (1968) How Committees Invent?. *Datamation, 14, 4, pp. 28-31*

[5] Freeman, P. and Hart, D. (2004) A science of design for software-intensive systems, *Communications of the ACM (CACM), 47, 8, pp. 19-21.*

[6] Hales, D. and Patarin, S. (2005) Computational Sociology for Systems "In the Wild": The Case of BitTorrent. *IEEE Distributed Systems Online, vol. 6, no. 7, 2005.* [DELIS-TR-0204]

[7] Hales, D. (2005) Emergent Group-Level Selection in a Peer-to-Peer Network. *Proceedings of the 2nd European Conference on Complex Systems, Paris, Nov. 2005.* [DELIS-TR-0200]

[8] Hales, D. and Arteconi, S. (2005) Friends for Free: Self-Organizing Artificial Social Networks for Trust and Cooperation. Submitted to IEEE Intelligent Systems Special Issue on Self-management through self-organization in information systems. Available: http://arxiv.org/abs/cs.MA/0509037. [DELIS-TR-0196]

[9] Kemerer, C. F. and Slaughter, S. (1999) An Empirical Approach to Studying Software Evolution, *IEEE Trans. Software Eng. 25, 4, pp. 493-509.*

[10] Mollona, E. and Hales, D. (2005) Knowledge-Based Jobs and the Boundaries of Firms. Accepted for publication in the Journal of Computational Economics. University of Bologna, Dept. of Computer Science Tech. paper UBLCS-2005-14. [DELIS-TR-0230]

[11] Solé, R. Ferrer-Cancho, R. Montoya, J. M. and Valverde, S. (2002) Selection, Tinkering and Emergence in Complex Networks, *Complexity, 8, 1.*

[12] Solé, R. and Valverde, S. (2004) Information Theory of Complex Networks, in *Networks: Structure, Dynamics and Function, Lecture Notes in Physics*, Springer-Verlag.

[13] Valverde, S. Ferrer-Cancho, R. and Solé, R. Scale-Free Networks from Optimal Design, *Europhysics Letters 60, pp. 512-517*

[14] Wagstrom, P., Herbsleb, J., Carley, K. (2005) A Social Network Approach to Free/Open Source Software Simulation. In Scotto, M. and Succi, G. (eds), *Proceedings of the 1st International Conference on Open Source Systems, Genova, 11th-15th July 2005.*

[15] Wasserman, S. and Faust, K. (1994) *Social network Analysis: Methods and Applications*, Cambridge University Press, Cambridge.