AA Distributed Software Development Methodology

Renato Fabbri*[1,3], Ricardo Fabbri†[2,3], Vilson Vieira‡[1,3], Alexandre Negrão[3], Lucas Zambianchi[3], Marcos Mendonça[3], Daniel Penalva[1,3] and Danilo Shiga[3]

[1]Physics Insitute at São Carlos (IFSC), University of São Paulo (USP), Brazil
[2]Polytechnic Institute (IPRJ), State University of Rio de Janeiro (UERJ), Nova Friburgo, Brazil
[3]LabMacambira.sourceforge.net

June 2, 2013

Abstract

We present a new self-regulating methodology for coordinating distributed team work called Algorithmic Auto-regulation (AA), based on recent social networking concepts and individual merit. Team members take on an egalitarian role, and stay voluntarily logged into so-called AA sessions for part of their time (e.g. 2 hours per day), during which they create periodical logs — short text sentences — they wish to share about their activity with the team. These logs are publicly aggregated in a Website and are peer-validated after the end of a session, as in code review. A short screencast is ideally recorded at the end of each session to make AA logs more understandable. This methodology has shown to be well-suited for increasing the efficiency of distributed teams working on what is called Global Software Development (GSD), as observed in our experience in actual real-world situations. This efficiency boost is mainly achieved through 1) built-in asyncrhonous on-demand communication, documentation of work products and processes, and 2) reduced need for central management, meetings or time-consuming reports. Hence, the AA methodology legitimizes and facilitates the activities of a distributed software team. It thus enables other entities to have a solid means to fund these activities, allowing for new and concrete business models to emerge for very distributed software development. AA has been proposed, at its core, as a way of sustaining self-replicating hacker initiatives. These claims are discussed in a real case-study of running a distributed free software hacker team called Lab Macambira.

*renato.fabbri@gmail.com
†rfabbri@iprj.uerj.br
‡vilson@void.cc

1 Introduction

One of the defining features of modern times is the widening geographical distribution of software teams [9] creating what is called Global Software Development (GSD) [6, 1, 2]. An example is the free software movement. Projects and institutions like Mozilla Foundation has several employees and thousands of voluntary developers distributed across many countries. The same is true for GNOME [6], OpenBSD, MySQL or Apache Software Foundation, to cite just a few of the most active projects.[1] Beyond the free and open software community, GSD has a growing popularity in every niche of the software industry as a whole, even among those distributing their software with proprietary licenses. This phenomenon is attributed to a variety of factors such as a larger labor pool, natural globalization of software companies and foundations or even the premise of cheaper cost of production [8].

Despite the advantages of GSD, we have noticed how difficult it is to coordinate and fund free software on a significantly larger scale than currently practiced, as series of qualitatively new situations arise. Distributed teams are very heterogeneous containing not only volunteers and very experienced developers, but also contractors and freelancers from different backgrounds and cultures. Our observations are founded on the factors suggested by Carmel [4] as main difficulties for GSD: distance, time and cultural differences. In the case of free or open software projects, all these factors are involved.

Another problem faced by modern software companies and other collectives are frequent ineffective meetings, which are seldom focused on the particular interest of any attendant. The result is that it has become the norm to participate in too many meetings with the "laptop open", which can be un-productive. Software developers like to code, to be productive, to have their hands on their project, to do what they are best at. They dislike to forcibly stop for meetings or to do other bureaucratic activities such as writing lengthy reports to justify their funding.[11]

In this paper we propose the AA methodology and an associated software system for coordinating distributed team work, tackling the disadvantages of GSD. Team members take on an egalitarian role, and stay voluntarily *logged* in the system for part of their time (e.g. 2 hours per day), during which they log a periodical short text sentence or *microlog* — similar to a 'tweet' from Twitter — as the status of their activity. Logging is carried out using a series of easy UI alternatives: unix shell commands, native GUI or web page, conventional social network posts, or chat messages to a log bot listening to IRC, GTalk, G+, and others. These "microblog sentences" are publicly aggregated and validated by other team members. Through AA, the community has a methodology and an associated system to help implement and validate the activities of a distributed software team. It implicitly legitimizes financial support for the expansion of the activity of a distributed development team. The AA methodology is specially useful for coordinating distributed and decentralized team work, providing

[1] Ohloh, the open source network, has a more complete and constantly updated list of the most active projects on-line at www.ohloh.net

effective means to asynchronously update different team members without the need for synchronous unproductive meetings.

A brief overview of current work on GSD methodologies related to AA is presented in Section 2. In Section 3 the most relevant characteristics of the AA methodology are outlined. In the same section, we report an actual use case of AA for coordinating a team of 9 paid developers during the second half of 2011, as well as a broader use case of AA from 2012 to the present time. In Section 5 we summarize the overall conclusions and indicate future possibilities for the practical use of AA in other types of teams of software developers or organizations working on non-software distributed activities.

2 Related work

There has been a large amount of research in methodologies to deal with distributed teams of developers. Although this paper focuses on GSD, some of its principles could also be adapted for smaller teams of developers working at the same place, timezone and with minor cultural differences, depending on the specific context and demands. Moreover, 'distributed development' is generally tought of as being global which is not true. For instance, AA has been effectively applied to a team whose members live in the same city but work at different timeframes and locations, see Section 4. Even smaller groups of developers working on the same building could use GSD methodologies (or an adapted subset) to their benefit, *e.g.*, to account for different work habits, minimize formal meetings, document work process and history, and so on. A thorough survey of these methodologies is beyond the scope of this paper; this section presents but a brief overview.

Various methodologies for GSD were built around the factors that affect distributed teamwork. As proposed by Carmel [4], these comprise three distances: geographical, cultural and temporal. First, geographical distance handicaps (*i*) *coordination*, the act of integrating all the tasks distributed between units [5]; (*ii*) *control*, or the process to maintain specific goals, policies or quality levels; and (*iii*) *communication*. All those factors are correlated, *e.g.*, a team needs to have clear communication to work on tasks of a specific problem.

Second, cultural distance encompasses differences on organizational and natural culture. Spoken language, unit and ethnic values are common forms of such distance. Some companies prefer to allocate development units in foreign locations with minimal cultural distance (*e.g.*, an American company may prefer Ireland due to spoken language similarity [5]). Third, the temporal distance that hampers synchronous communications like telephone or videoconferences. Units of developers working on different time-zones are concerned with managing of their agenda guided by this temporal distance.

Targeting geographical distance, Carmel [5] suggests a strategy to reduce intensive collaboration. His approach divides the whole software life-cycle into levels of complexity. Each level has a degree of collaboration. For example, some developers working on a project with high collaboration level should use

the follow-the-sun approach: when concluding the work day, they pass their work to the team working on another time-zone. Other tactics are suggested by the same author to deal with the three distances, such as separating foreign units of developers in time-zone bands.

Battin *et. al.* [3] propose and discuss their experiments using specific methodologies created for the distributed development centers from Motorola (at the time having 25+ software development centers worldwide). These methodologies included constant communication with critical units, incremental integration and schedules based on time-zones distributed to developers on 6 countries from 3 continents.

In considering free software projects instead of companies, similar factors are present and specialized methodologies arise. German [6] provides a concise review of the methodologies used by the GNOME project, one of the most active of all free software projects. The manuscript is centered on the software architecture. It begins by explaining that GNOME is separated into modules (76 on version 2.4, to be precise) and each module has one maintainer who divides her modules into separate parts in which other developers can work on independent tasks, along other responsibilities. As in other free software projects, all development was done using a *bugtracker* for bug and issue management, mail lists and Internet Relay Chat (IRC) for discussion and communication and a version control system like Git or Mercurial. Periodical (commonly yearly) conferences like GUADEC is common on free and open source projects for face-to-face meetings and is based on a different place each time.

3 The AA methodology

Some of the strategies for GSD mentioned on the previous section are based on complex methodologies and many were built for a specific company or software center. We propose an alternative methodology based on a simple idea: small working sessions logged by a computational tool. Figure 1 summarizes this methodology.

3.1 The AA session

From the developer's perspective, the AA methodology is based on generating small, high level reports called *micrologs* or *AA shouts* of what they are doing in a specific period of time. This timeframe can be between 5 to 15 minutes in our proposed practice, depending on what is most convenient for the developer and the team. An *AA Session* is a focused period of continuous work, lasting about 2 hours in our proposed regime, in which the developer issues a collection of these short periodical *micrologs*. The developer can set reminders or alerts to show up when it is time to *microlog*. The objective of the flexible timeframe and alert scheme is to *minimize developer overhead* during his AA session. In this way, the developer can issue micrologs while staying maximally focused on his code. Each microlog may be sent directly to an on-line server, or stored

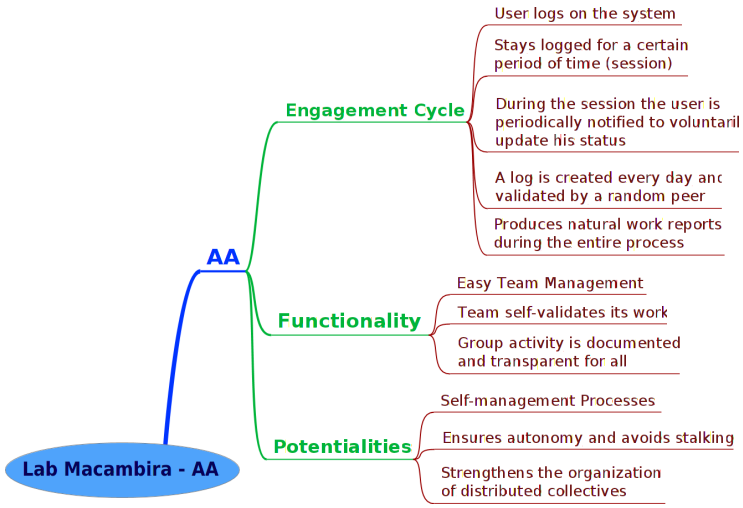

Figure 1: A mindmap of the AA methodology: 1) 'developer engagement cycle', *i.e.*, his use of AA; 2) 'functionality', *i.e.*, main goals of the system; 3) 'potentialities', *i.e.*, enhancements AA delivers to the developer's context.

locally in a temporarily database for sending/pushing later on. This enables offline micrologging and periodical alerting.

Developers optionally record a brief *video screencast* at the end of the session summarizing what has been done, explaining his goals and challenges in his own words and showing his most important results. This is very similar to the video logging system in the movie Avatar, although it is clear from our July 2011 git repositories and online wiki that we have used this powerful concept in AA prior to the release of Avatar. Moreover, screencasting differs from general videologging in that it typically captures actual workflow on the computer screen. Screencasting, combined with the textual log of the AA Session, renders the final report more understandable for the individual developer himself or to other people searching for information about his production.

3.2 The AA website report

All AA reports made by the developers are ultimatelly sent to a Web server and become publically aggregated on a Website called pAAnel. It is then possible for a manager or another developer to follow very closely the work of a given developer, nearly real-time, reading each of his small reports or micrologs of what he is working on.

Another possibility is to check older sessions to check when certain tasks were carried out and the comments of the developer about the *process*. Since each AA microlog happens in a very short timeframe of work, the information about what was was done – specially *how* it was done – becomes very easy to

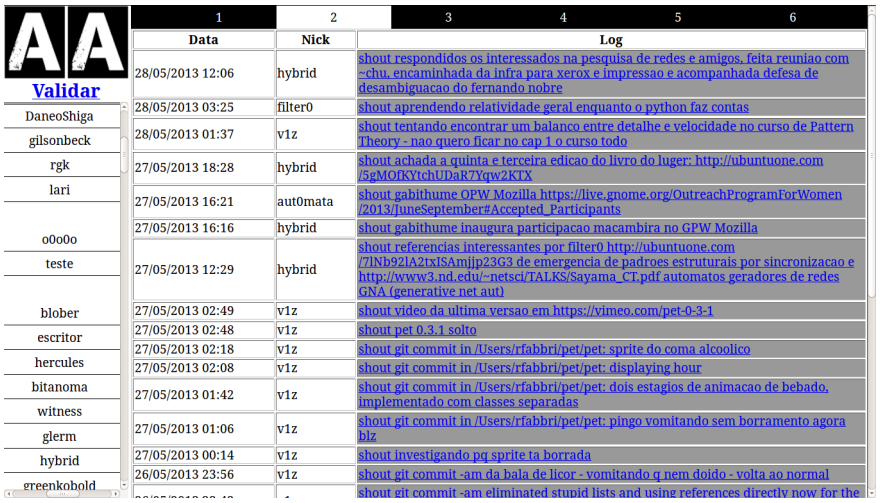

Figure 2: AA Report Agregator Version 0.1. Messages of users hybrid, filter0, v1z e aut0mata about activities of interest for labMacambira.sf.net and a diversity of collaborating entities (IPRJ/UERJ, IFT/UNESP, IFSC/USP, OPW/Mozilla, PulaPirataComics). Each message is a "shout", which, grouped, forms an AA session.

understand, as opposed to having a long report in the end of a session.

In the current version of the AA server infrastructure, the aggregating website allows the developer to attach a link for his screencast about each session. Screencasts are specially useful on cases where the small reports were done in a hurry, because the developer did not want to lose his focus on something critical at that moment.

3.3 Peer validation

No set bosses or leaders are required in an ideal aplication the AA methodology; in practice, the role of bosses basically disappears or is greatly alleviated – hence the name Algorithmic *Autoregulation* and other implicit interpretations to the AA acronym and symbol. The primary mechanism to achieve AA is peer coordination and social behavior, be it deliberate or implicit. In order to prevent spamming and to improve the overall quality of AA reports, each AA session must be validated by another developer. More specifically, all reports are read by someone that will mark them collectively as 'valid' or 'invalid' and may optionally write commentaries about the specific session and quality of micrologs. The developer in charge of validating any given session is randomly assigned by the AA Web server, which sends an email to the chosen developer with an URL to a validation interface.

Peer validation also helps in making decentralized collaboration more cohe-

sive by encouraging members to be minimally aware of peer activities, even when these are not immediately useful for accomplishing the task at hand. We have observed that decentralized teamwork can get so efficient at actual production that the team gets short-sighted in terms of coordination: non-communicating subteams can get formed in practice if care is not taken, causing a fragmentation of the collective. Peer validation is one way to help avoid fragmentation and is an essential mechanism of decentralized team autoregulation.

4 Results and discussion

Easy and effective GSD team management is the main purpose of the AA methodology. We applied the proposed methodology to a group of 9 developers in July of 2011, 3 of which are coauthors of the present work, on what we called Lab Macambira.[2] The main objective of the team was to work on an array of strategic free software projects in the broad audiovisual and web categories, contributing directly to their development, submitting bug patches or committing new features to their source code.

The team members had different levels of experience on software development for large and distributed free software projects like Scilab or Mozilla. In this way, one month of training was conducted by three experienced developers (the first authors of the present work), teaching the basics of use of development infrastructure tools like bugtrackers, programming languages, version control systems, and build systems. After this period, a starter project was proposed for new developers: to submit a bugfix or implement a new feature and have an accepted patch or commit to the official repository for a large free software project. Developers passing the starter project would be deemed 'initiated' and be called a 'Macambira' developer, and be hired for the remainder of the semester. Table 1 summarizes the effective accepted contributions of each successfully initiated developer to free software projects in 2011, which used the AA methodology.

In one month, each developer officially contributed to one or many free software projects. Many developers started the initiation training with no knowledge of what is free software and ended that period becoming a free software developer. During that month, the same team of trainees also developed the first version of the AA system and used AA to manage their activities, even while developing the other aforementioned free software projects. Thus, AA and the associated software system was tested, prototyped, and developed in close contact with actual practice. The source code of AA — both the client that sends the logs and the AA Web server — is public available as free software[3] and all the actual AA log data of the entire team of Lab Macambira from 2011 to the present time is also on-line[4].

[2] LabMacambira.sf.net: http://labmacambira.sf.net

[3] AA source code: http://labmacambira.git.sourceforge.net/git/gitweb.cgi?p=labmacambira/aa

[4] Logs of AA sessions: http://labmacambira.git.sourceforge.net/git/gitweb.cgi?p=

Table 1: Free and open software projects that received contributions from successfully initiated developers or 'Macambiras' that worked under the AA methodology. The first column lists applications to which contributions were officially accepted and whose development process was tracked using AA. The second column shows the pseudonym of the committers. At Lab Macambira it is common practice to use pseudonyms in AA as identification in order to enhance privacy.

Application	"Commiters"
Mozilla Firefox	daneoshiga, bzum
Evince	hick209, bzum, marcicano, mquasar
BePDF / Xpdf	marcicano
Ekiga	flecha
Empathy	fefo
Lib Folks (Telepathy)	kamiarc
Scilab	v1z, humannoise
VxL	v1z
ImageMagick	v1z
OpenOffice	hick209
Puredata	v1z, automata, greenkobold, gilson, bzum
Puredata OpenCV	v1z
Puredata GEM	v1z, fefo, hick209
Puredata PDP	v1z, fefo, hick209
ChucK	rfabbri, automata
ChucK MiniAudicle	rfabbri, automata
WebRTC	automata
OSC-Web	automata
Web-PD-GUI	automata
Live-Processing	automata
ChucK-Wiimote	automata
Audiolet	automata
Extempore	automata

After the initial training period of 1 month, the initiated 'Macambiras' worked during 6 additional months on a large range of free software projects, divided into work groups — each work group focusing on a specific theme like video, audio and web — and funded by contracts, freelance, and support of the Pontão Nós Digitais[5]. Table 2 has a list of *new* free software applications created by 'Lab Macambira' since July 2011.

While using the AA system, developers learned to work asynchronously with others and got used to the habit of periodically updating their status on their projects. Each programmer was given the chance to work with considerable freedom, in any place and time of preference. The strictest required responsibility was that of using AA for at least one sesssion of 2h per day, while working on the agreed upon tasks. The online pAAnel allowed each developer to quickly grasp activities from others while avoiding interrupting them, a process further aided by the screencasts. Adjustments to the task deadlines and milestones (which were managed in Trac) were done based on verified progress of individuals and labMacambira.sf.net team as a whole. The various newcommers benefitted from a fast learning team by the use of AA as a flexible and simple transparency system. Updates from the team were not transmitted on a person-to-person basis, but rather on a person-to-team basis based on the available online progress information.

As of this writing, 'Lab Macambira' unites over 15 software developers, and key developers trained in 2011 continue to work voluntarily in the project.

5 Conclusions

In a scenario where Global Software Development is growing as a popular form of software development in the entire software industry, there is an increasing need for methodologies to deal with its potential disadvantages and at the same time to amplify its advantages.

This paper has presented a new methodology for GSD, in scenarios involving a series of large or small groups of software developers, either working on different countries or at the same room. The AA methodology implements a simple system where each developer take notes of his work posting a periodic log of small text sentences or micrologs. The sum of those activity logs, along with an entire session of work, results in a complete unit of report. The report is made public available through a Website and is validated by peers that are randomly selected by the AA Web server.

AA is not limited to a work-management tool, but acts as *a methodology to improve the time sensibility of individuals*, helping divide their complex tasks in time into small chunks or sessions, and also reducing the need of extensive reports or unnecessary meetings. By asking users to write a minimal text sentence as a continuous log feed, the proposed methodology avoids disturbing the

`labmacambira/paainel`

[5] http://nosdigitais.teia.org.br

Table 2: Software projects created by Lab Macambira since July 2011 with a short description and the technologies involved (*e.g.*, programming languages or frameworks). It is interesting to note the heterogeneity of projects and their areas of application.

Application	Description	Technologies
AA	Algorithmic Auto-regulation	Python, PHP
Ágora Communs	System for on-line deliberations	PHP
SIP	Scilab Image Processing toolbox	C, Scilab
Animal	An Imaging Library	C
TeDi	Test Framework for Distance Transform Algorithms	C, Shellscript, Scilab
Macambot	Multi-use IRC Bot	Python
Conferência Permanente	Platform for the permanent conference of the rights of minors	PHP, JavaScript
CPC	Center for accounting of the Brazilian culture representation groups	Python, Django
Timeline	Interactive time lines on the Web	JavaScript
Imagemap	Interactive marking for on-line photos	JavaScript
ABT	Program for real-time execution and musical rhythmic analysis	Python
EKP	Emotional Kernel Panic	Python, ChucK
SOS	Aggregation and diffusion of popular and native knowledge about health	Python, Django
Creative Economy	Platform for creative, collaborative and solidarity economy of the culture hubs and cultural entities	Python, Django
OpenID Integration	Adaptations to existing software for unified login through OpenID	PHP
pAAinel	Dashboard for the real-time visualization of Lab Macambira activity	Python, Django
Georef	Collection of scripts to be used as reference, which aims to be a GIS platform to map public data of use to citizens	Python, Django
AirHackTable	Software for an instrument which generates sound from flying origami tracked by webcams	Puredata, C++, Scilab

flow of developers which are heavily concentrated on programming: developers just have to type a few words and go back to coding.

We have analyzed data and gained practical experience on the use of AA to auto-regulate the work of Lab Macambira, a group of free software developers from Brazil. Since July of 2011 the group has contributed and created new free and open source software for a vast number of additional applications.

The AA methodology is not restricted to software development, even though it was designed for the latter. As of this writing there is an entertainment studio, Pula Pirata[6], that has been using AA to manage their creative activities. Other people with no software background, like social scientists, musicians and activists also have been using AA and contributing for its broader improvement.

There are many aspects of the work which remain unfinished. Additional ubiquitous client interfaces for micrologging from different interfaces beyond IRC, *e.g.*, web social services and email, would greatly make the use of AA easier and more widespread, turning AA a truly ubiquitous and replicable system, presented on everyday communication channels. Another research direction is that the actual work logs generated by the Lab Macambira and Pula Pirata collectives since July of 2011 could be statistically analyzed aiming to recognize patterns in the behavior of individuals and their creative production process. It would also be desirable to provide more extensive experiments and psychological studies focusing on further backing specific claims made in this paper, in order to refine the methodology, its mechanisms and parameters.

Acknowledgments

The authors acknowledge the financial support from Pontao Nos Digitais, and Ricardo Fabbri acknowledges support from FAPERJ/Brazil 111.852/2012. We would also like to thank AA: the present research and even this manuscript was written using AA. The complete log is on-line at www.pulapirata.com/skills/aa.

References

[1] Global software development and delivery: Trends and challenges. 2

[2] Global software development: Who does it? 2

[3] Battin, R., Crocker, R., Kreidler, J., and Subramanian, K. Leveraging resources in global software development. *Software, IEEE 18*, 2 (2001), 70–77. 4

[4] Carmel, E. *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999. 2, 3

[6] www.pulapirata.com

[5] Carmel, E., and Agarwal, R. Tactical approaches for alleviating distance in global software development. *Software, IEEE 18*, 2 (2001), 22–29. 3

[6] German, D. M. The gnome project: a case study of open source, global software development. *Software Process: Improvement and Practice 8*, 4 (2003), 201–215. 2, 4

[7] Gobbo, F., and Vaccari, M. The pomodoro technique for sustainable pace in extreme programming teams. *Agile Processes in Software Engineering and Extreme Programming* (2008), 180–184.

[8] Komi-Sirviö, S., and Tihinen, M. Lessons learned by participants of distributed software development. *Knowledge and Process Management 12*, 2 (2005), 108–122. 2

[9] Last, M. Understanding the group development process in global software teams. In *Frontiers in Education, 2003. FIE 2003 33rd Annual* (2003), vol. 3, IEEE, pp. S1F–20. 2

[10] Reis, C. R., and de Mattos Fortes, R. P. *Caracterizaç ao de um Processo de Software para Projetos de Software Livre*. PhD thesis, University of São Paulo, Brazil, 2003.

[11] Thompson, C. Solo performance: shut up and start acting like an introvert. *Wired 20.04* (April 2012), 036. 2