AA Distributed Software Development Methodology

Renato Fabbri Ricardo Fabbri Vilson Vieira
Alexandre Negrao Lucas Zambianchi Marcos Mendonca
Danilo Shiga

August 8, 2012

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, as in code review. A video is generally recorded by the members to make their work logs more understandable. This methodology is well-suited for increasing the efficiency of distributed teams working on what is called Global Software Development (GSD) through asynchronous on-demand communication, reducing the need for central management, unproductive meetings or time-consuming reports. The AA methodology also legitimizes the activities of a distributed software team. It thus enables 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 as a methodology is also at the core of having self-replicating hacker initiatives. These claims are discussed in a real case-study of running a distributed free software hacker team called Lab Macambira.

1 Introduction

One of the defining features of modern times is the widening geographical distribution of software teams [6] creating what is called Global Software Development (henceforth GSD) [4]. An example is the free software movement. Projects and institutions like Mozilla Foundation has many employees and thousands of voluntary developers distributed around many countries. The same is true for GNOME [4], OpenBSD, MySQL or Apache Software Foundation, to enumerate

just the most active projects[1]. Along the free and open software projects, GSD is growing popularity in every niche of software industry as a whole, even on those distributing their software with proprietary licenses. This phenomenon is attributed to various factors like larger labor pool, natural globalization of software companies and foundations or even the premise of cheaper cost of production [5].

However, dispite the advantages of GCD, we have noticed how difficult it is to coordinate and fund free software on a larger scale than currently available, when teams are very heterogeneous containing not only volunteers and very experienced developers, but also contractors from different backgrounds and cultures. Our observations found base on the factors suggested by Carmel [2] as main difficulties for GCD: 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 to the interest to any attendant. The result is that it has become the norm to participate in too many meetings with the "laptop open", which is unproductive at the very least. 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 have to stop for meetings or to write lengthy reports to justify their funding.

Here we propose AA, a methodology and an associated software system for coordinating distributed team work dealing with 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 — similar to a "tweet" in this Twitter era — as the status of their activity using an easy to use command. These "microblog sentences" are publicly aggregated and validated by other team members. Through AA, we have a methodology and an associated system to validate and enable the activities of a distributed software team. It implicitly legitimizes financial support for the expansion of the activity of the developer team. The AA methodology is specially useful for coordinating distributed and decentralized team work, providing effective means to asynchronously update different team members without the need for synchronous unproductive meetings.

A brief overview of current papers about GSD methodologies related with AA is presented in Section 2, while in Section 3 we describe the most relevant characteristics of the AA methodology followed by the description of our experience using AA in a team of 9 developers since July of 2011. Finally, in Section 5, we draw some final conclusions and indicate future possibilities to the practical use of AA on other teams of software developers or individuals working on non-software activities.

```
[ A very good article on the value of asynchronous communication for personal and
group productivity, related to the key necessity of having moments of introversion
to avoid daily pressures of forced socialization.  The way we work on the digital age
```

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

enables people to be very productive, the article also mentions linux as a hallmark example [7]]

[TODO: cite CIA.vc bot stuff]

2 Related Work

There has been a large amount of research done in the area of methodologies to deal with distributed teams of developers. We are focusing in GSD here, however some principles involved on those methodologies could be used on smaller teams of developers working in the same place, time and with minor cultural differences. Moreover we generally think on "distributed development" being global which is not totally true. We even applied AA to a team that work at the same city but on different times (more details on Section 4). Even smaller groups of developers working on the same building could use GSD methodologies. A thorough survey of these methodologies is beyond the scope of this paper. Here we present a brief overview.

Various methodologies for GSD were built around the factors that affect distributed team works, proposed by Carmel [2] and comprising three distances: geographical, cultural and temporal. Geographical distance prejudice *coordination*, being the act of integrating all the tasks distributed between units [3]; *control*, or the process to maintain specific goals, policies or quality levels; and *communication*. All these factors are correlated, for example, a team need to have clearly communication to work on tasks of a specific problem. Cultural distance encompass differences on organizational and natural culture. Spoken language, unit and ethnic values are common forms of this distance. Some companies prefer to situate development units in foreign locations with minimal cultural distance (e.g. an American firm prefer Ireland, because of spoken language similarity [3]). And finally the temporal distance that prejudices 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 [3] suggests a tactic to reduce intensive collaboration. His approach divides the whole software life-cycle on 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, like separating foreign units of developers in time-zone bands.

Battin et al. [1] propose and argues about their experiments using specific methodologies created for the distributed development centers of the Motorola Company (which has over 25 software development centers worldwide). These methodologies included constant communication with critical units, incremental integration and scheduled based on time-zones distributed to developers on 6 countries from 3 continents.

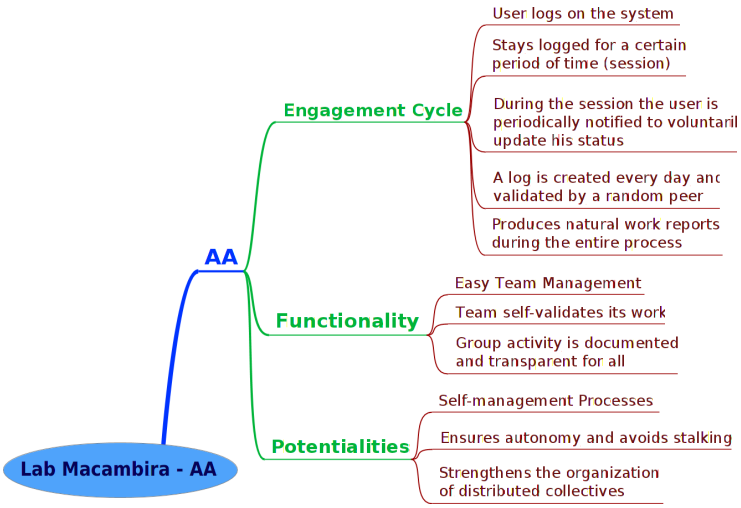

Figure 1: Mindmap of our methodology

While considering free software projects instead of companies, the same factors are present and some methodologies arises. German [4] gives a concise review of methodologies used by the GNOME project, one of the most active free software projects and used by companies like Sun Microsystems. It is interesting to note a difference on viewpoint present on his paper: German focuses the methodology description on code. He start explaining that GNOME is separated into modules (76 on version 2.4, to be precise) and each module has one maintainer who divide his modules into separated parts in which other developers can work on independent tasks, along other responsibilities. As like others free software projects, all the development was done using a *bugtracker* to bugs and issues management, mail lists and Internet Relay Chat (IRC) to discussion and communication and a software configuration management like SVN or Git. Periodical (commonly yearly) conferences like GUADEC is common on free and open source projects to face-to-face meeting and is based each time on a different place.

3 The AA Methodology

As noted, some strategies for GSD is based on complex methodologies and many of those were built for a specific company or software center. Here we propose an alternative methodology based on a simple idea: small working sessions logged by a computational tool. Figure 1 summarizes our methodology.

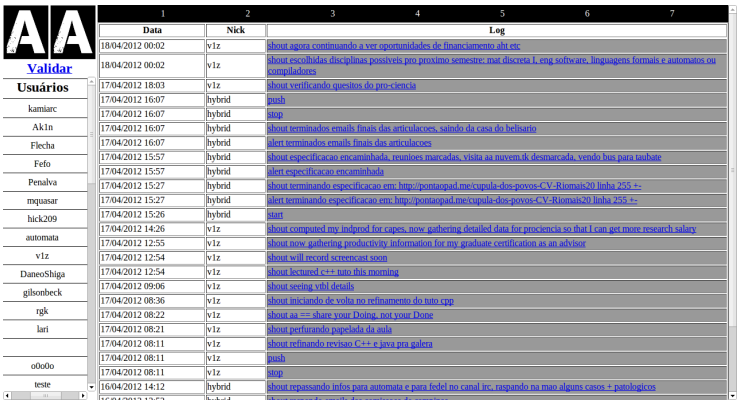

Figure 2: AA Version 0.1

3.1 AA Session

From the developer perspective, the AA methodology is based on creating pretty small high perspective reports of what they are doing in a specific time frame, that can be something between 5 to 15 minutes, depending in what is more comfortable for the developer. An AA Session would be a period of at least 2 hours doing these reports. The developer can set reminders to show up when its time to make a new report. The objective of the flexible time frame and reminders is to minimize developer overhead during his AA session. In this way he can make the reports while staying concentrated in his code. Each report can be sent directly to an on-line server, or stored locally in a temporarily database for sending later. This make possible the use of AA while offline.

Developers can also record a video screencast in the end of the session summarizing what has been done, explaining with his words and showing his most important results. This, combined with the textual log of his session, makes the whole report more understandable for himself or other people searching for information about his production.

3.2 AA Website Report

All AA reports made by the developers are sent to a Web server and become public in a Website, being possible then for a manager or another developer to follow very closely the work of a developer, nearly real-time, reading each of his small reports of what he is doing right now.

Another possibility is to check older sessions to see when sometimes was done and the comments of the developer about it, since each AA post happens in a very small time frame of work, the information about what was done become very easy to understand, instead of a long report in the end of a session.

The site is where the developer can add to his session a link for his screencast

about that session, a video summary of what was done to complement the reports, useful specially on cases where the small reports were done in a hurry, because the developer did not want to lose his focus on something important at that moment.

3.3 Validation

Each AA session must be validated by another developer, it means that all reports are read by someone that will consider then valid or not and will even write commentaries about the specific session. The developer in charge of validating a session is decided randomly by the AA Web server, which send an email to the chosen developer with an URL to a validation interface.

4 Results and Discussion

The easy and effective management of teams working on GSD is the main purpose of the AA methodology. We applied this methodology to a group of 9 developers in July of 2011 on what we called Lab Macambira [2]. The main objective of the team was to work on different free software projects, contributing directly to their development, sending bug corrections or proposing new features on their source code.

All the team members had different levels of knowledge on software development as part of large and distributed free software projects like Scilab or Mozilla. In this way, one month of training was conducted by two experienced developers, teaching the basics of use of development support tools like bugtrackers, programming languages and version control systems. After this period, a challenge was proposed for the new developers: send a bug correction or a new feature to a large free software project and you will work with us, you will be a "Macambira" developer. Table 1 summarizes the contributions of each "Macambira" to free software projects.

In one month, each developer contributed to many very large free software projects. Many of the developers started the training with no knowledge of what is free software and ended that period becoming a free software developer.

During that month, the same team developed the first version of the AA system and used AA to manage their activities. Even while developing the system. All the source code of AA — the client that sends the logs and the AA Web server — is public available [3] and all the AA sessions log of the whole team of "Lab Macambira" is also on-line [4].

After the training period, during more 6 months, the "Macambiras" worked on a large range of free software projects, distributed on work groups — each

[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=labmacambira/paainel

Table 1: Free and open software projects that received contributions from "Macambiras". On the first column we can see a list of applications of those projects. At right, the pseudonym of the "Macambiras" who sent *commits* to the application. At "Lab Macambira", and at free software community in general, is a common practice to use pseudonym as identification.

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

work group focusing on a specific theme like video, audio and web — and financed by contracts and support of the "Pontão Nós Digitais". In Table 2 we can see a list of the free software created by "Lab Macambira" since July of 2011.

As of this writing the "Lab Macambira" have many software developers, and some of the trained developers continue to work voluntarily in the project.

5 Conclusions

In a scenario where GCD is growing as a popular form of software development, not just on free software projects but in the whole software industry, we need methodologies to deal with its disadvantages and at the same time to amplify its advantages.

This paper has presented a methodology to GCD, being the development conducted on large or small groups of software developers, working on different countries or even at the same room. The AA methodology implements a simple system where each developer take notes of his work generating a periodical log of small text sentences. The sum of those sentences, along an entire session of work, results in a complete report. The report is made public available through a Website and be validated by other developers sorted aleatory by the AA Web server.

Instead of a merely work-management tool, AA act as a methodology to improve the time sense of individuals, dividing their work on small sessions, and also reducing the need of extensive reports or unnecessary meetings. By asking users to write a minimal text sentence as a continuously log, AA does not disturb developers concentrated on programming: developers just have to type some characters, hit *enter* and go back to coding.

AA application is not restricted to software development. As of this writing there is a comic book studio [5] starting to use AA to manage their activities. People with non-software background, like social scientists, musicians and activists has also using AA and contributing for its improvement.

For developer teams, we have experienced 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 have contributed and created new free and open source software for a vast number of applications.

There are many aspects of the work which remain unfinished. New ways to report logs — the "Twitter like" messages — from different interfaces like IRC, Internet Messaging services and email can make the use of AA easy and widespread, turning AA an ubiquitous system, presented on everyday communication channels. Even the work logs generated since July of 2011 could be vastly statistically analyzed aiming to recognize patterns in the behavior of individuals and their productions.

We would like to conclude setting an important role of AA: being a free software system and an open methodology, AA could be used to auto-manage

[5] Pula pirata: http://pulapirata.com

Table 2: Software projects created by "Lab Macambira" since July of 2011 with a short description and the technologies — like programming languages or frameworks — involved. It is interesting to note the heterogeneity of projects and its areas of application.

Application	Description	Technologies involved
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/C++, Scilab

groups of individuals working on software or other kinds of activities. In this way, we are interested to spread AA for those groups, to have even more developers contributing in a collaborative way.

Acknowledgments

We would like to also thanks AA: this whole research and even this paper was written using AA. The complete log is on-line at http://hera.ethymos.com.br:1080/paainel/casca/.

References

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

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

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

[4] 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. 1, 4

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

[6] 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. 1

[7] Thompson, C. Solo performance: shut up and start acting like an introvert. *Wired 20.04*, 2 (April 2012), 70–77. 3