Humanizing Software Artifacts: Software Engineering with Intelligent and Social Software Artifacts as our Friends

Mithun P. Acharya
mithunacharya.com

Abstract—We propose and explore a new paradigm that considers every software artifact such as a class as an artificially intelligent and a socially active entity. In this Software Artifact Choreographed Software Engineering (SACSE) paradigm, the humanized artifacts themselves take the lead and choreograph (socially, in collaboration with other intelligent software artifacts, humans, and artifact/human organizations) software engineering solutions to the many software development and maintenance challenges such as (automatic) code reuse, documentation, testing, patching, and refactoring. In this paper, we discuss the implications of seeing software artifacts as our new intelligent friends. (Abstract)

Index Terms—Software Artifact Choreographed Software Engineering (SACSE), Artificial Intelligence, Social Network Analysis, Cloud Services (key words)

I. INTRODUCTION

Millions of classes sit idly for years in the version control systems doing nothing but change to the whims and fancies of the developers – they are treated as mere objects (pun intended). Well, no more. Enter a new world where classes, our new human-like friends, are highly social creatures. Classes form facebook-like social networks with their fellow classes and humans, tweet, email, instant message, and even apply for IPv6 address(es) and make phone calls. These classes take great pride in their design and correctness and even have linkedin-like profiles to boast about their achievements. The classes, apart from being social, are highly intelligent as well. For example, classes learn, reason, and exercise their freedom of tweet/speech and democratic voting rights in deciding how they, or an API they use, evolve or get refactored. Also, like humans, classes spend a lot of time browsing the social network and the Internet to proactively and continuously find opportunities for self-improvement; as an individual and also as a socially responsible community member. In doing so, classes choreograph efficient solutions to many software development and maintenance challenges such as reuse, documentation, testing, patching, and refactoring. Later, driven by human-like tendencies, classes religiously publish their improvements and accomplishments on their linkedin, webpage, and social network. These social and intelligent classes, in millions, will take the lead and join forces with humans in creating better and reliable software of the future.

The rest of the paper is organized as follows. In Section II, we explore a few initial directions for implementing the SACSE paradigm – we discuss how SACSE complements or can build upon existing ideas, tools, and practices in Software Engineering. In Section III, we discuss some of the implications of implementing the SACSE paradigm on all things Software Engineering. We discuss related literature in Section IV. Finally, we conclude in Section V.

II. THE WAY FORWARD

Implementing a platform for humanizing software artifacts and enabling SACSE will require a concerted and sustained engineering efforts and cross pollination of ideas from areas as diverse as Artificial Intelligence, Social Networks Analysis, Cloud Services, Big Data Analytics, Natural Language Analysis, and last but not the least, many areas of Software Engineering including empirical studies on developer preferences and behaviors [26][28], Mining Software Repositories [17], and Automated Software Testing in practice [20][29]. In this section, we attempt to provide some initial directions for implementing and enabling the SACSE paradigm.

Enabling SACSE will not require any changes to the classes themselves. However, the classes will need a platform to go social and will have to subscribe to a series of services that make them (artificially) intelligent. In the subsequent paragraphs, we discuss the initial ideas for implementing such a social platform for collaboration and a service platform for intelligence.

Repositories such as GitHub [7], Q&A sites such as StackOverflow [14], and Codebook [4] can be viewed as social networks of people and software artifacts. In GitHub and StackOverflow, the code artifacts the developers produce or discuss are at best second class citizens. In Codebook, artifacts are passive entities that do not actively participate like humans do in people social networks such as Facebook. By promoting classes to the first class status and enabling the classes to actively participate (harnessing the services offered via the service platform, discussed next), these social networks can be extended to serve as an early prototype that implements the social platform for the SACSE paradigm. The envisioned social platform overlays the developer, code, and organization dependencies/networks and lets humans and classes harvest the synergies among them.

The service platform will enable classes (who have their primary residence in repositories such as GitHub) to subscribe to services implemented on the cloud. For example, making a Boa-like infrastructure [21] available to each class as services on the cloud enables all classes to search ultra-large scale

---

1 A software artifact is any tangible by-product of software development such as requirements documents, UML diagrams, classes, and test cases. While the ideas discussed in this paper may apply to any software artifact, we discuss our ideas with a class (a code artifact) representing a software artifact.
repositories and ask a range of questions [2]. With google and facebook-like search services, classes will be able to harness the power of the Internet and their social network of developers and other classes. Likewise, software paradigms inspired from the world of living organisms such as mutation testing [10], clone detection [11], genetic programming [5], search-based software engineering [15][22], and automatic repair [23] are a natural fit for SACSE and can all be implemented as cloud services, readily available to the classes. Along with the aforementioned software engineering and search services, classes will also have access to AI services such as learning, reasoning, speech recognition/synthesis, and Watson-like [9] natural language analysis through the cloud. Classes can then be seen as powerful autonomous agents with self-improvement goals. Each class will analyze itself, ask questions on its social network, reason searching the Internet, and then proactively use the aforementioned search and software engineering services for self-improvement, concretely achieved through refactoring for performance, or testing [20] and patching for correctness. All these enable these humanized classes to enter the realm of artificial intelligence and make optimal, and at times, greedy choices that are best for themselves. In the next section, we discuss the many scenarios that are made possible in a world that is SACSE-enabled.

III. IMPLICATIONS

Let us, for now, assume that the SACSE paradigm is fully implemented and deployed. Like the public Internet and company-specific intranets, we will have a single public SACSE platform for all things open source and private SACSE platforms that are company specific. However, unlike the public Internet, let us assume that the public SACSE platform will be free, unregulated, and somehow democratically managed by humans and classes.

Classes tweet and send friend requests to developers, who have the experience to fix or improve them if needed. Developers follow the classes they developed and also the classes they are interested in. Developers no longer have to look for classes to work on or fix next – the classes that are in need contact relevant developers directly. In a company, a group of existing related classes might directly contact a product manager after these classes read the requirements gathered by the manager from a customer interview, published on the company Intranet. These classes will directly negotiate requirements with that customer from that point on and keep the traceability links between requirements, code, and test up to date.

Classes add their parent, sibling, and children classes into their family network. Packages, libraries, team, and products will all have fan pages which the classes and developers may like. The evolution history of a class will be published to its timeline. Classes write recommendations on the linkedin page of their peer classes they have come to trust over time. Classes may also unfriend lazy classes that are forever bug-prone. With all these, the complex relationships and dependencies among developers, software artifacts, and organization structure, that are often deeply buried or lost in source repositories (and hence, a subject of interest to the MSR [17] researchers), will be intuitively evident and browsable by classes and humans alike. The large corpus of class and developer tweets and the network itself lends nicely to social network analysis [13] and advanced network and tweet visualizations [16] with graph analytics [8].

With Boa, google and facebook-like search available as services, classes constantly search stackoverflow, national vulnerability databases, problem discussions in the consumer forums, hacker exchanges, and repositories on the Internet to auto-fix themselves and alert other relevant classes/developers on their social network through a private message or a phone call, if required. With search, software reuse will no longer be a major problem for companies. With the classes from different teams in the company being “aware” of each other because they are proactive and share the same company-wide social and service platform, reuse opportunities will be discovered in near real time and informed to the relevant stakeholders.

IDEs as we know today such as Visual Studio and Eclipse will evolve from just being smart editors to virtual conference rooms in the social network. Developers and other classes will meet in such conference rooms and collaborate to discuss and compose new software through touch screens [25] and gestures. Editing code will become a thing of past. The classes, harnessing the power of their social and cloud platforms, will be able to converse and work with the developers using Siri-like interfaces [1] during the otherwise mundane sessions such as debugging and refactoring.

IV. RELATED WORK

Codebook is a platform that uses a social-networking inspired approach to connect artifacts and people in software repositories [4]. However, in Codebook, artifacts are at best passive entities that broadcast their updates to the social network newsfeed only when humans change them. People networks such as Facebook largely owe their success to the fact that most people are actively participating entities. People browse facebook, observe the world, search the web, introspect, then they change and improve, and tweet and get on with life – and the cycle continues. In this spirit, unlike Codebook, with the SACSE paradigm, artifacts are active entities – they are intelligent, they think for themselves, and they have improvement goals just like us humans. Classes achieve their improvement goals by leveraging the software engineering, search, and AI services accessed via SACSE’s service platform. To further compare the SACSE paradigm and the Codebook platform, consider the example given in Section 1 of one of the Codebook publications [18]. Program manager Klaus sifts through the newsfeed in sort.DLL’s codebook homepage and concludes that his team’s upcoming product release will be delayed. sort.DLL is just one library dependency and Klaus' project might have hundreds of DLL dependencies. In practice, Klaus will not have time to keep up with all the DLLs. With SACSE, sort.DLL will “herself” proactively infer (hence, the term “artifact choreographed”) that Klaus’ team will not meet the upcoming release deadline and notify Klaus over a private message or a phone call. This inference requires that sort.DLL has natural language analysis and company intranet search
capabilities and SACSE provides these capabilities to sort.DLL through its service platform over the cloud.

Crowdsourced approaches exist for variety of Software Engineering tasks ranging from documentation [27] and design to coding [24], debugging [3], and testing [6]. With a social platform such as the extended GitHub or Codebook, SACSE nicely complements crowdsourced Software Engineering in that it brings together humans, software artifacts, and human/artifact organizations into a single platform and channelizes the crowd efforts and the class-choreographed initiatives into a single energy stream.

In the Social Internet of Things (SIoT) [12] paradigm, billions of embedded computing devices form a social network. However, such social networks exist for humans to provide a structure to the Internet of Things (IoT) and efficiently navigate and access the results of the social inter-device communication. In SACSE, unlike SIoT, classes, like humans, constantly seek opportunities for improvement using the power of the service platform and then publish such improvements to its social network. In doing so, classes choreograph efficient solutions to many software development and maintenance challenges such as reuse, documentation, testing, patching, and refactoring. In the distant future, it will not be surprising to see the seamless merging of the present day Internet, SIoT, and the social platform of SACSE.

V. CONCLUSIONS

In this paper, we proposed and discussed a new paradigm called Software Artifact Choreographed Software Engineering (SACSE) that sees every class as an artificially intelligent and a socially active entity. The key idea is to provide a platform for classes to go social with developers and their peer classes and making the power of search, software engineering, and AI available to each class as cloud services. Object-oriented programming (OOP) allowed encapsulation and information hiding. With OOP, we can have modular objects which own their data and are responsible for own behavior. With SACSE, we go a step further and require that attributes such as correctness, security, and performance (targets for self-improvement) of a class should be its responsibility too – not just data and behavior. So, these socially responsible classes take a lead in improving themselves and also the communities they are part of, choreographing efficient software engineering solutions along the way. The three key attributes of the SACSE paradigm are:

1. Artifacts are intelligent and they are active social entities.
2. In addition to behavior and data, the socially responsible classes are also responsible for their improvement as an individual and a community member.
3. Software Engineering solutions are artifact-choreographed with little or no human assistance. These solutions are tied to the improvement goals of the individual classes.

Someday the SACSE paradigm might liberate developers and maintainers from mundane activities such as refactoring for performance, finding opportunities for reuse in a company, testing, and following security best practices. Instead, developers will be able to focus their energies on creative activities such as, well, creating more creative classes/citizens. We provided some initial directions for realizing the vision of a SACSE-enabled world. And in envisioning such a world, let us hope that these classes, our new friends, will be socially responsible, devoid of all dark and evil emotions.

REFERENCES

[1] Apple Siri. apple.com/ios/siri/.
[2] Boa. Boa.cs.iastate.edu
[3] BugFinders. bugfinders.com.
[4] Codebook. research.microsoft.com/en-us/projects/codebook/
[5] CREST open workshop on Genetic Programming for Software Engineering. http://crest.cs.ucl.ac.uk/cow/28/.
[6] crowdsourcedtesting. crowdsourcedtesting.com.
[7] GitHub. github.com.
[8] Graphlab. graphlab.com.
[9] IBM Watson. ibm.com/smarterplanet/us/en/ibmwatson/.
[10] International Workshop on Mutation Analysis. sites.google.com/site/mutationworkshop2014/.
[11] International Workshop on Software Clones. softwareclones.org
[12] Social Internet of Things. social-iot.org.
[13] Social network analysis. en.wikipedia.org/wiki/Social_network_analysis.
[14] StackOverFlow. stackoverflow.com.
[15] Symposium on Search Based Software Engineering (SBSE). ssbse.org
[16] Twitter visualizations. interactive.twitter.com.
[17] Working Conference on Mining Software Repositories (MSR). msrconf.org
[18] A. Begel and T. Zimmermann. Keeping up with your friends: function Foo, library Bar.dll, and work item 24. Web2SE 2010.
[19] C. Bird, T. Menzies, and T. Zimmermann. The art and science of analyzing software data: Analysis patterns. Elsevier. 2015.
[20] E. Bounimova, P. Godefroid, and D. Molnar. Billions and billions of constraints: Whitebox fuzz testing in production. ICSE 2013.
[21] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. BOA: A language and infrastructure for analyzing ultra-large scale repositories. ICSE 2013.
[22] M. Harman. The current state and future of search based software engineering. FoSE at ICSE 2007.
[23] D. Kim, J. Nam, J. Song, and S. Kim. Automatic patch generation learned from human-written patches. ICSE 2013.
[24] K. R. Lakhani, D. A. Garvin, E. Logstein, “TopCoder: Developing Software through Crowdsourcing,” Harvard Business School Case 610-032, 2010.
[25] S. Muller, M. Wursch, T. Fritz, and H. C. Gall. An approach for collaborative code reviews using multi-touch technology. CHASE at ICSE 2013.
[26] E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. ICSE 2009.
[27] C. Parnin, C. Treude, L. Grammel, and M. A. Storey. Crowd documentation: Exploring the coverage and the dynamics of API discussion on Stack Overflow. ICPC 2013.
[28] Y. Tao, J. Kim, S. Kim, and C. Xu. Automatically generated patches as debugging aids: A human study. FSE 2014.

[29] N. Tillmann, J. Halleux, and T. Xie. Transferring an automated test generation tool to practice: From Pex to Fakes and Code Digger. ASE 2014