Collaborative Software Development with GitHub® Projects

Preprint · July 2018

DOI: 10.13140/RG.2.2.19507.27681

3 authors:

Marius Golombeck
University of Applied Science and Arts Dortmund
20 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Henning Orlowski
University of Applied Science and Arts Dortmund
8 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Stefan Tübben
University of Applied Science and Arts Dortmund
12 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Collaborative Software Development with GitHub® Projects

Marius Golombeck, Henning Orlowski, Stefan Tuebben

July 14th 2018

University of Applied Sciences Dortmund
Department of Computer Science
Emil-Figge-Straße 42, 44227 Dortmund, Germany
{magol003, heorl001, sttue002}@stud.fh-dortmund.de

Abstract: Following the ongoing trend of physically detached teams in collaborative software development, we examine the current possibilities of collaborative systems. At first we give an introduction to collaborative systems and the process of software development. Afterwards we analyse common problems that occur during collaborative software development. As a possible solution for the aforementioned problems regarding collaborative software development, we introduce GitHub® Projects, a functional, methodological approach towards a structured development process.

Keywords: Collaborative Software Development, GitHub®, GitHub® Projects

1 Introduction

1.1 Topic Description

As software engineering projects grow bigger and tend to emphasize the collaborative work of individuals with different professional backgrounds, the need for standardized work principles increases [1].

A possible structured approach to this process is the usage of collaborative software during the development process [2]. The code hosting and collaborative development platform GitHub® provides an interface and a workflow that takes up common problems in the collaborative development process [3]. In this paper we examine this workflow from a theoretical point of view and discuss possible problems of collaboration.

While a growing number of corporations perceive collaborative software development as a positive influence in modern software engineering projects, finding a standardized solution is an ongoing problem [4]. Collaborative software development is not only a question of finding proper technical solutions, but seeing it's tasks as social-technical challenges in general [5, 6].

1.2 Approach and Structure

We will start with the basics of collaborative systems and lead on to a methodical approach for the introduction of collaborative systems and the issues that it entails, especially regarding collaboration and project management. Later on we present GitHub®, a collaborative coding platform, that aims to solve the aforementioned issues. Finally, we sum up our findings and give an outlook.

2 Research

2.1 Collaborative Systems

Collaborative systems evolve today, among other things, from the growing need for interpersonal communication and rules of physically detached teams in today's projects [7]. While these systems potentially magnify the chance of success for projects, problems such as lacking acceptance among the participants are common and can lead to poor project results [8].

Moreover alarming from a methodical point of view is the lack of social research regarding collaboration-intensive software development and its associated socio-technical issues [9]. The identification of these problems beforehand is critical, as well as a thorough understanding of the whole collaborative software development process [10]. Collaborative systems can help identify and solve these problems, as they support standardized already proven work processes [11].

The term collaborative system is often associated with so called groupware, as they are in fact synonyms as the following definition from the early 1990's indicates: "Collaborative systems or groupware are computer-based systems that support groups of people engaged in a common task (or goal) that provide an interface to a shared environment." [12]. Following this definition, you can see the emphasis on people as opposed to more technical definitions, that are often found among classic single-user software.

2.2 A Methodical Approach for the Introduction of Collaborative Systems

During the introduction of collaborative systems, a lack of acceptance can threaten the success of a whole system change in general [13]. Hereinafter we will discuss a possible approach to solve problems during the introduction of these systems. While it is still tempting to simply order the usage of a new process top-bottom, experiences in the past indicate higher chances of success, if a nuanced methodical approach is used for their introduction [14]. The following rough methodology is inspired by the general procedure known as change management [15], recapitulated by our interpretation for the most important aspects during the introduction of a new collaborative system (see Figure 1) [15].

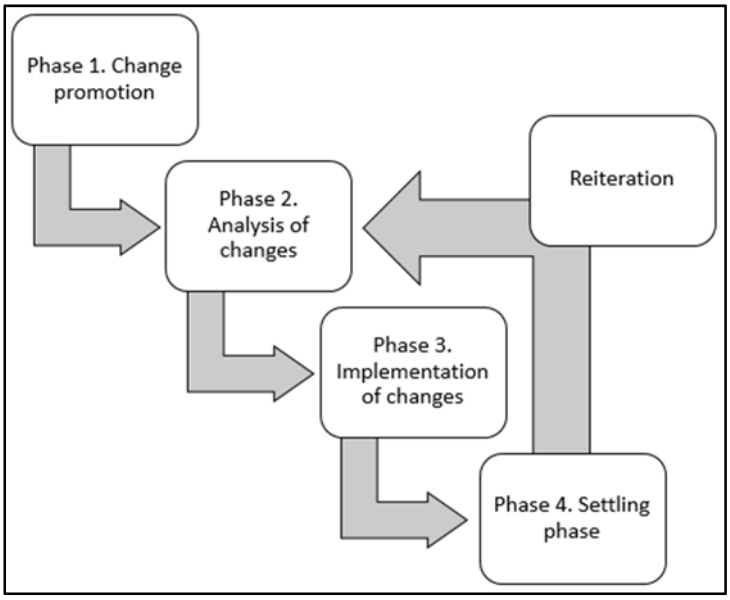

Figure 1 - Phases recommended for the introduction of collaborative systems[1]

Step 1: First of all, you should promote the project goals and objectives of this methodical change [15]. Research indicates that the rate of acceptance rises when employees are involved early in the change process, so they have a feeling of participation and the possibility to understand the need for the system change [15, 16]. Employee tend to see effects on their known workflow negatively, therefore a strategy of obscurity may lead to a further fundamental attitude of rejection [17]. Once a more favourable degree of acceptance has been reached, the change should be executed as quickly as possible [15]. Strategies like a parallel adoption[2] or big bang[3] seem intuitionally superior to an incremental adoption[4] in this regard.

Step 2: Once the implementation of this change has been completed, a phase of analysis is advised [15]. The identification of the changes, their effect on the projects culture and processes, followed by a corporate discussion with the affected employees could lead to improvement of the initial concept [15]. While the possibilities of changes are rather small through the usage of standard software [20], even the rejection or approval of small possible variations within the workflow could lead to a sense of common responsibility, as opposed to another rise of shadow IT[5] [22]. If specific steps of the process are skipped, its effort doesn't justify the observed benefits.

Step 3: The implementation of these variations bring along costs and impacts as well as the need for a plan of specific introduction [15]. These aspects need to be defined, changes scheduled and quickly executed similar to the introduction of the initial collaborative

[1] Adapted from [15]

[2] The implementation of a new software system while running and using the old system in parallel is referred to as parallel adoption. [18]

[3] The term big bang is borrowed from the introduction of new enterprise resource planning systems and means the implementation of a new software system in one single go. [19]

[4] As opposed to the parallel adoption, the incremental means the gradually replacement of an old software system with a new system. [18]

[5] Shadow IT is the use of IT-related hardware or software by a department or individual without the knowledge of the IT or security group within the organization. This phenomenon is often associated with the existence of insufficient official processes within an organization and a loss of control. [21, 22]

system [15]. Changes that lead to an unviable project need to be rolled back and changes with a positive effect should be taken to the next iteration of changes [15].

Step 4: The change or introduction of central new collaborative system demands above-average commitment of all participants in addition to tasks in the day-to-day operations [23]. This has to be a phase of settling, where neither scheduled analyses, further changes or planned discussions should take place. In favour of long term beneficial working environment, phases of lesser organizational stress should be integral [24].

After the first four steps have been successfully completed, a phase of constant reiteration of steps two, three and four, should be implemented in the projects culture [15]. As requirements of a project may shift throughout development phases, even once positively perceived processes could become a threat for the project's success.

2.3 Common Issues of Software Development

As the trend of decentralised software development continues to grow [25], new concepts regarding cooperative work need to keep up with the phenomenon of global work. Common problems within this phenomenon can be classified in six distinct dimensions [26].

At first we identify general strategic issues. In this regard new requirements are difficult to implement into the whole project structure, as soon as the original scope of the project has been set [26]. Solutions for collaborative system need to take this into account and use social-technical models and mechanisms to solve these conflicts.

Secondly, cultural issues are the main driving forces for interpersonal conflicts within a project [26]. Since a lot of cultural backgrounds with different needs and communication styles collide constantly, personal resentments grow and influence the productivity within the project team enormously [26].

Similar, but not identical to the second issue, is the problem of inadequate communication [26]. Especially during the early development phases the amount of communication needed to clarify the scope, destination, approach and details of a project is too high to use virtual forms of communication efficiently instead of a local meeting [27]. While this problem usually changes as the project progresses to downstream development stages, it's still both difficult and important to find the right form of communication for specific circumstances [28].

Another neglected problem within collaborative development processes is knowledge management [26]. Decision processes for complex problems in projects take precious time [29]. It is critical for a successful undertaking, that the results of former endeavours are well documented, specifically addressed and easily accessible [27]. The constant repetition of otherwise already settled matters does not only demotivate every participant but threatens the transparency and therefore the chances of success for a given project [27, 30].

Project and process management issues appear when the members of a team lose their common perception of a projects status. This is also known as desynchronization [26]. For example, physically detached teams interpret milestone definitions differently. As group "A" could interpret the milestone of a working module as an alpha tested working module, the group "B" would interpret the very same milestone as only finished when beta tests have successfully been conducted. Even this seemingly trivial example could

lead to significant delays during the production process and the unavoidable delay of subsequent partial projects on a critical path, as soon as these too modules are joined.

Typical for the complex environment of software development processes are technical issues, that could appear due to the innovative character of the developed software or because of production related issues, that appear within a network of decentralised developers [26]. A possible example would be unreliable and slow connections within the project network or incompatibility of development environments [31].

3 GitHub® Collaboration Tools

3.1 Introduction to GitHub®

GitHub® is a code hosting platform for version control and collaborative software development. It was founded in 2008 and has since grown to be the largest hoster of source code in the world [32]. GitHub® uses the features of the version control system Git, as well as adding their own features, such as Bug Tracking, Issue Management, Project Management, Wikis and Statistics. In June 2018 Microsoft has announced it will acquire GitHub® for $7.5 billion [33].

The basic functions of GitHub® are as follows:

Code: The key function of GitHub® is Code Management. The code-page can be considered an overview of your projects code. Here you are able to manage source code, commits, branches, releases and contributors.

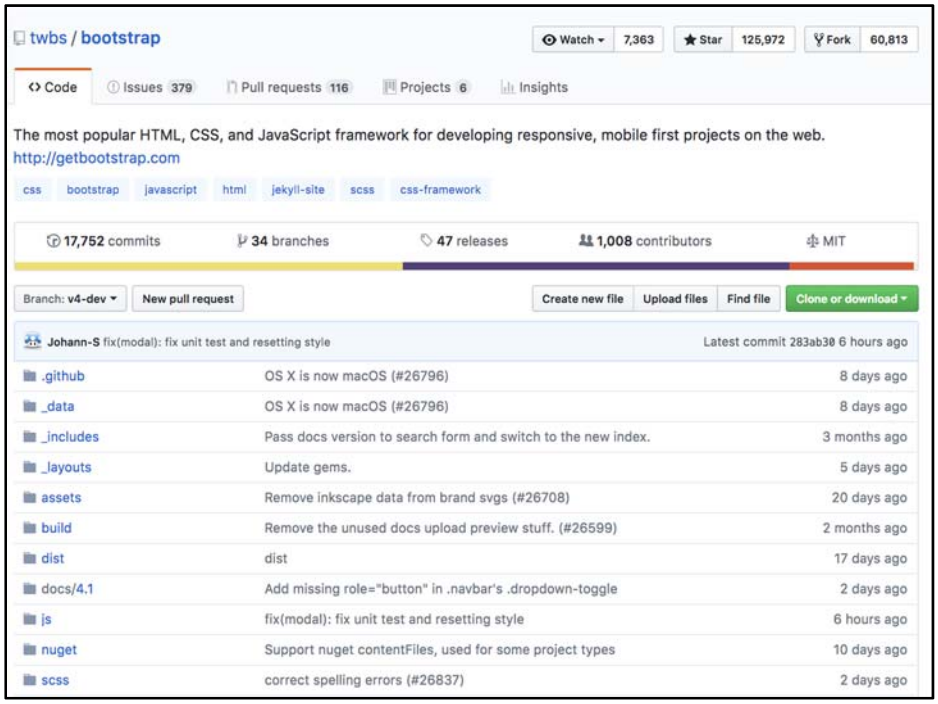

Figure 2 - The code-page of Bootstrap, showing various core elements of GitHub® [6]

Issues: Issue Management is another key element in modern software development projects. The issue-page is not only designed to be used to manage issues, but also to create and follow tasks and to-dos. You are able to create labels, assign issues to projects

[6] Source: https://github.com/twbs/bootstrap

and/or developers, as well as create milestones. Powerful filter options allow you to maintain an overview of the project development status.

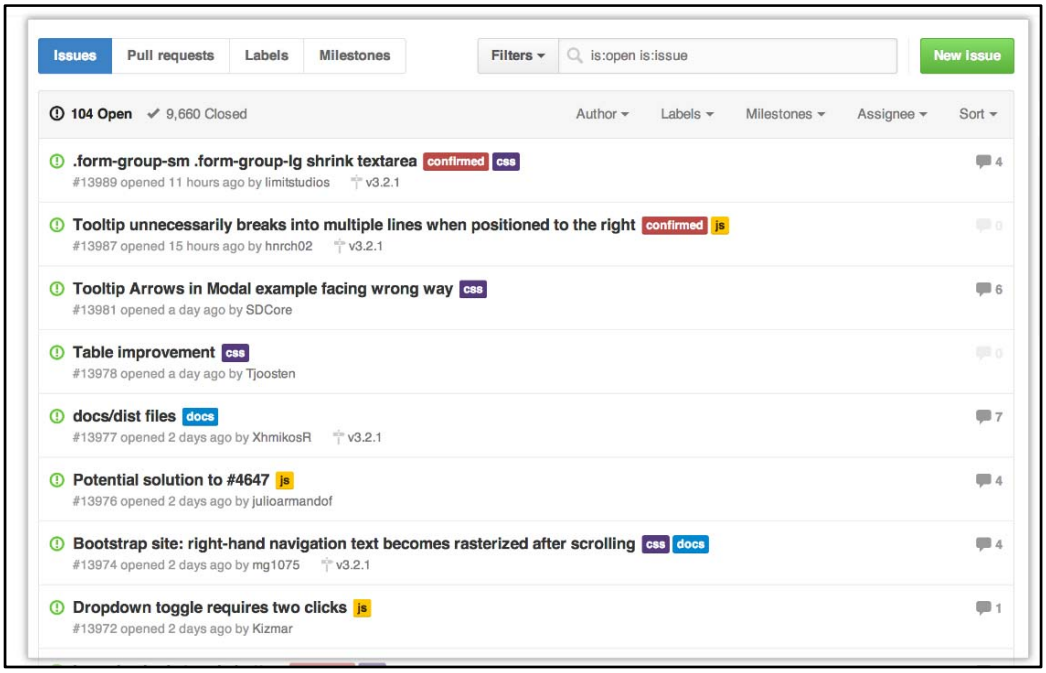

Figure 3 - The issues-page of Bootstrap [7]

Pull Requests: The concept of pull requests originates from Git, where it represents a user (you) requesting another developer from the repository (e.g. the project admin) to pull a branch from your repository to their repository.

Imagine two repositories and two branches. The developer, who is contributing to the feature branch now requests the admin of the official project repository to pull his branch (feature). Comparing a pull request to a simple merge, a pull request has the advantage that it initiates a discussion around the content that is staged to be pulled, before it gets integrated into the main codebase.

Projects: See 3.2.

Wiki: In GitHub® wikis are being used to create and host software documentation. While you are able to use the comment function from commits, or the comment function from issues to annotate your code in a quick and simplified way, you are able to write an extended documentation of your project here. Keeping the documentation in close alignment to the code helps maintaining a software project and keeping it up-to-date.

[7] Source: https://guides.github.com/features/issues/

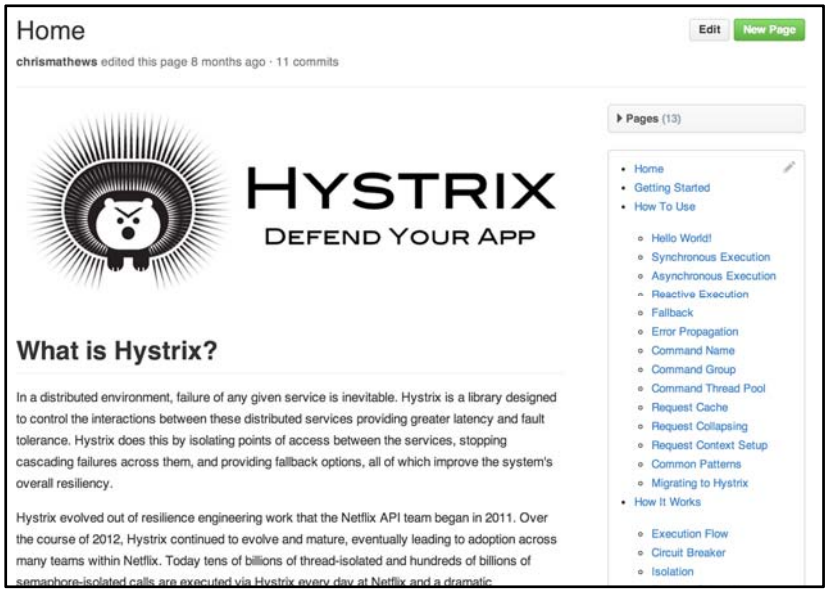

Figure 4 - An example of the Wiki function of GitHub® [8]

Insights: Insights are representing statistics of your projects. It contains statistics of contributions, merges, networks, dependencies or commits. Especially for larger projects with insights you are able to generate reports and gain an overview from a more statistical perspective.

Figure 5 - The insights-page of GitHub® [9]

3.2 Structured Development with GitHub® Projects

Software Development today is rarely done by a single person anymore. Development in small teams, especially in agile environments has become the industrial standard over the past couple of years.

For an efficient development process project management and software development need to be connected as tightly as possible.

[8] Source: https://help.github.com/articles/about-github-wikis/
[9] Source: https://github.com/twbs/bootstrap/pulse

GitHub® Projects has been introduced in October 2016 for organisations on GitHub® and has since been made available for regular users. The idea is to "manage projects in the same place you keep your code" [34].

You start by creating a new project where you are able to select a project template.

GitHub® Projects focuses heavily on the Kanban software development methodology for project management and improving interaction. It has originally been thought up by Taiichi Ohno, an engineer at Toyota with logistics and manufacturing background, whose aim it was to improve manufacturing efficiency [35]. Kanban literally translates to "visual signal" and has since found manifold application, especially in software development and project management [36].

Kanban uses "cards" to manage work items. Every work item is represented on a separate card and every team member is able to view them on the project board and follow progress.

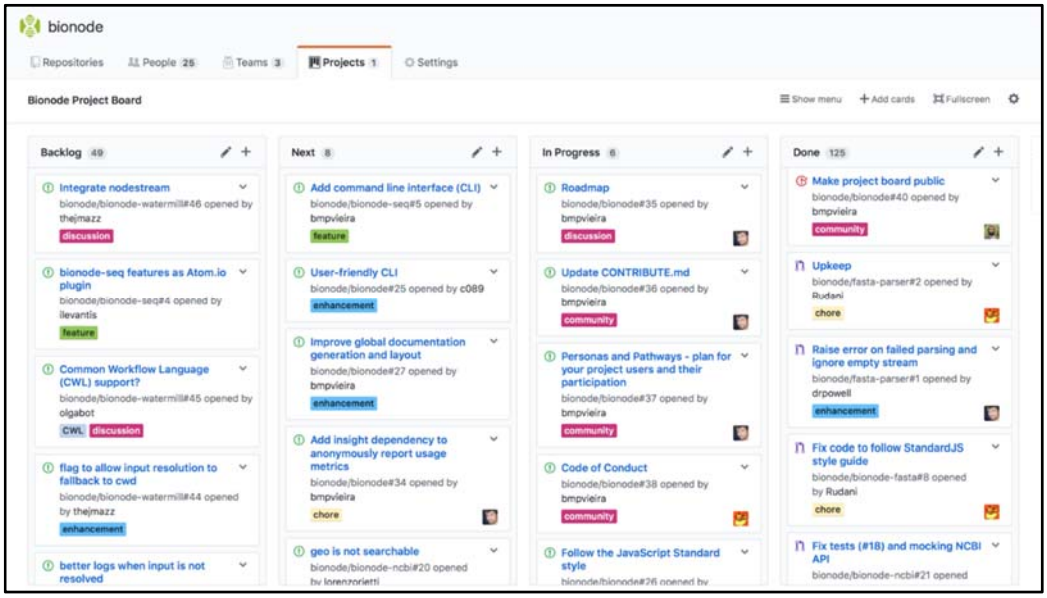

Figure 6 - Example showcase of a kanban board in GitHub® Projects [10]

In GitHub® Projects you are able to choose Kanban templates, like basic or automated (with or without reviews), as well as bug triage or start without a template. These pre-configured templates help you get a head start in your project.

After creating a new Project - with or without the support of a template - you are being offered the following functions.

[10] Source: https://medium.com/@bmpvieira/how-to-make-your-github-organization-project-board-public-fb5a243f28e4

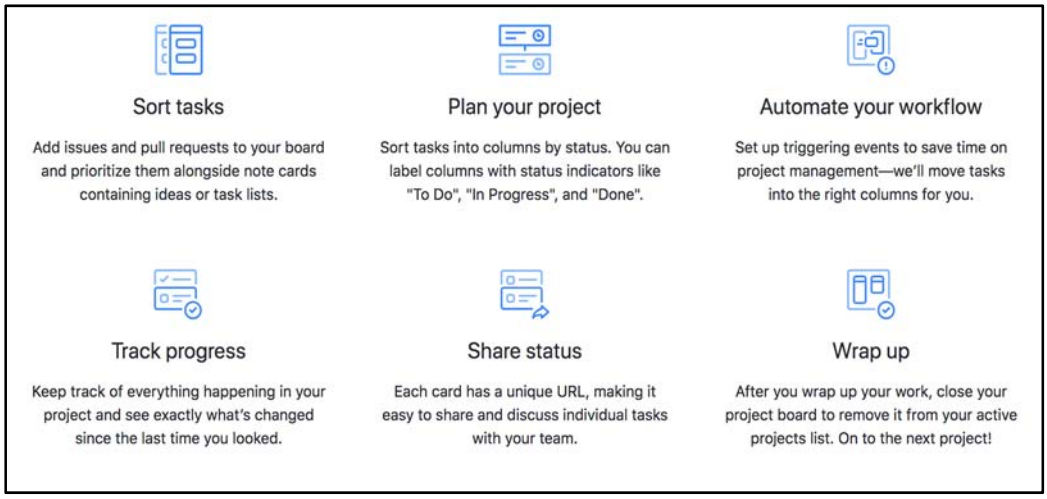

Figure 7 - GitHub® Projects functions [11]

Task Management: Any type of work can become a project task. Take issues, bug reports, to-do lists or ideas as examples. Task Management includes project management related functions, like being able to prioritize, sort and assign work like you would in project management.

Project Planning: Project Planning introduces the element of time to project management. Tracking time allows you to track progress and thus sync project management and software development. Particularly with regard to the development status, GitHub® Projects supports the user with status indicators, such as “To Do”, “In Progress”, or “Done”.

Workflow Automation: Project Management is a complex and tedious process. It can be simplified by using workflow automation. GitHub® Projects offers the ability to set up triggering events. This leads to minimizing planning mistakes and reducing the issue probability in mission critical elements, as their lifecycle has been automated.

Tracking Progress: In addition to the aforementioned points of Task Management and Project Planning you are able to keep track of the projects progress in its entirety.

Status Sharing: Many software development projects include working with people abroad, or at least not in the immediate vicinity. Every card on the project board contains a unique URL so the cards can be shared in the project team. Furthermore, the cards can be discussed in the team and these discussions can be introduced to the project themselves.

Wrap Up: Often overlooked and not deemed as important as the aforementioned elements, the wrap up process, in the ending stages of the project, plays a large role in project management today. Taking the lessons learned from a project, and having a clear closing line helps with future projects. Additionally, past projects are being stored on GitHub® Projects, so that in future projects you will be able to look back and see exactly what has happened, even if its lodged deep in the project details.

3.3 Social Coding

Social Coding is a software development approach that aims towards a collaborative environment for developers, offering possibilities of sharing, discussing, automating and

[11] Source: https://github.com/squasseK/PorscheShuttleTracker/projects

planning code or projects. Due to the paradigm shift towards a more agile development approach, this idea has become more and more common in the past couple of years. To evaluate the increase in popularity it is appropriate to compare the social coding approach to a non-collaborative team development approach as well as to a lone development approach. We must first define these three different approaches [37, 38].

Social Coding: Social, agile, team-driven development approach with active, keen communication between project members.

Non-collaborative Team Development: Independent, loosely connected team approach to development with lesser communication between project members

Lone Development: Single person development in a barely collaborative, or non-collaborative environment.

We now compare these three approaches with the following criteria we have determined to be suited best for comparison.

Level of Collaboration: Measures the level of possible collaboration from factors like code sharing and issue management.

Agility / Reaction Speed: Measures the reaction speed to structural project changes.

Transparency: Measures the transparency of the development status. Note that transparency is largely dependent to personal responsibility.

Manageability: Measures the level of manageability on a personal / organisational level.

Synchronization with project management: Measures how connected the development process is, compared to the project management.

	Lone Development	**Non-collab. Team Develop.**	**Social Coding**
Level of Collaboration	n/a	Low	High
Agility / Reaction Speed	High	Low	Medium - High
Transparency	Medium*	Low*	High*
Manageability	High	Low	Medium - High
PM Sync Level	Medium*	Low	High*

Table 1 - Comparison of development approaches*[12]

We can sum up that lone development - as it contains no levels of direct collaboration - is the fastest to react to structural project changes, as well as the easiest the manage. However, the project management sync levels and transparency are largely dependent on the person doing the development.

[12] Transparency and project management synchronization depend largely on the personal responsibility

Non collaborating teams struggle in terms of agility, manageability and project management sync. They do have the ability to complete a higher workload, as they are a team after all. However, the collaboration issues will cause problems, especially with project management sync.

In theory, teams utilizing social coding reach the highest level of collaboration. The level of agility is medium, but depends largely on the team size. While being easier to handle than non-collaborating teams, collaborating teams are harder to manage than lone developers. The project management sync is high when utilizing proper social coding instruments, which also guarantee a high level of transparency.

4 Conclusion

We have described the basics of collaborative systems and shown the common issues in software development. Afterwards we presented GitHub® as a collaboration platform for software development. We compared lone development with non-collaborative team development and team development with social coding emphasis.

Our findings are that the shift towards collaborative forms of software development is justified, as a collaborative, social approach towards software development projects with a close connection to project management allows for a fast reaction time, high transparency, and good manageability. GitHub® provides an excellent base for collaborative development and is therefore suggested by us for small and large scale development projects.

However, the project members must follow the agile project management approach and actively communicate with each other to collaborate well. Only then the team is really able to benefit from the collaborative software development idea and the project can thrive.

5 Outlook

As the basic technologies of artificial intelligence and machine learning mature, the amount of companies that are able to use these technologies continues to grow. In our opinion these technologies will play a major role in the future of collaborative software development.

Starting with the possibility of better real time translations, that might help international teams to use synchronous forms of communications while using their own language to express their thoughts, might help tremendously during the discussion of complex problems. Another great possibility lies within the further automation of basic programming tasks, such as the generation of interfaces between modules of simultaneously working developer groups or the generation of test units.

The great benefit of an automated approach in this regard, is that the team is able to focus on the project on a higher level without losing precious resources during the development of easily standardized code brackets. This way the customer experience centric approach of agile development is emphasized, development cycles can be shortened and the quality through the usage of machine learning algorithms and automatically conducted test procedures improve.

Despite these practical improvements during the development process other trends like the continued growth of outsourcing endeavours and self-employed developers will lead

to a further rise in physically detached software development teams. Extrapolating these trends, more and more employees will work from home with a whole new range of demands for collaborative systems, since the motivation of employees in non-bureau environments tend to be harder to begin with.

References

[1] S. Sahay, B. Nicholson, and S. Krishna, *Global IT outsourcing: Software development across borders*. Cambridge: Cambridge University Press, 2003.

[2] J. DeFranco-Tommarello and F. P. Deek, Eds., *Proceedings of the 35th Annual Hawaii International Conference on System Sciences // Collaborative software development: Abstracts and CD-ROM of full papers : 7-10 January, 2001[sic], Big Island, Hawaii // A discussion of problem solving models and groupware technologies*. Los Alamitos, Calif: IEEE Computer Society Press, 2002.

[3] GitHub®, *Understanding the GitHub® Flow.* [Online] Available: https://guides.GitHub®.com/introduction/flow/. Accessed on: July 13th, 2018.

[4] ICSE; Institute of Electrical and Electronics Engineers; Association for Computing Machinery; IEEE/ACM IEEE International Conference on Software Engineering; International Conference on Software Engineering, *2016 IEEE/ACM 38th IEEE International Conference on Software Engineering companion - ICSE 2016: 14-22 May 2016, Austin, Texas, USA : proceedings*. Piscataway, NJ: IEEE, 2016.

[5] Institute of Electrical and Electronics Engineers; IEEE International Conference on Global Software Engineering; ICGSE, *2015 IEEE 10th International Conference on Global Software Engineering (ICGSE): 13 - 16 July 2015, Ciudad Real, Spain ; 2006 - 2015: 10 years of ICGSE*. Piscataway, NJ: IEEE, 2015.

[6] E. Ulich, "Arbeitssysteme als soziotechnische Systeme – eine Erinnerung," in *Journal Psychologie des Alltagshandelns*, pp. 4–12.

[7] S. Gera, "Virtual teams versus face to face teams: A review of literature," *IOSR-JBM*, vol. 11, no. 2, pp. 1–4, 2013.

[8] L. Falk and A. Eriksson, "How to develop usable groupware," Graduation paper, Teknisk- naturvetenskaplig fakultet UTH-enheten, Uppsala University, Uppsala, 2010.

[9] T. Hildenbrand, F. Rothlauf, M. Geisser, A. Heinzl, and T. Kude, "Approaches to Collaborative Software Development," in *2008 International Conference on Complex, Intelligent and Software Intensive Systems*, Barcelona, Spain, 2008, pp. 523–528.

[10] R. Saad, *Challenges and Issues in Collaborative Software Developments*, 2011.

[11] P. Rubens, *Why you should (sometimes) let software run your business.* [Online] Available: https://www.cio.com/article/3189142/application-development/why-you-should-sometimes-let-software-run-your-business.html. Accessed on: July 13th, 2018.

[12] C. A. Ellis, S. J. Gibbs, and G. L. Rein, "Groupware - Some Issues and Experiences," in *Communications of the ACM*, pp. 39–58.

[13] V. Kumar, B. Maheshwari, and U. Kumar, “An investigation of critical management issues in ERP implementation: Emperical evidence from Canadian organizations,” *Technovation*, vol. 23, no. 10, pp. 793–807, 2003.

[14] P. Madkan, “Empirical Study of ERP Implementation Strategies-Filling Gaps between the Success and Failure of ERP Implementation Process,” in 2014.

[15] S. Nagpal, S. K. Khatri, and A. Kumar, “Comparative study of ERP implementation strategies,” in *2015 Long Island Systems, Applications and Technology*, Farmingdale, NY, USA, 2015, pp. 1–9.

[16] S. Zimmermann, C. Rentrop, and C. Felden, “Managing Shadow IT Instances - A Method to Control Autonomous IT Solutions in the Business Departments,” in *AMCIS*, 2014.

[17] Cisco, “What is Shadow IT?,” [Online] Available: https://www.cisco.com/c/en/us/products/security/what-is-shadow-it.html. Accessed on: July 13th, 2018.

[18] N. Harramach, “Der Versuch einer Organisationsveränderung gegen die Organisationskultur,” in *Unternehmenskultur in der Praxis*, J. Herget and H. Strobl, Eds., Wiesbaden: Springer Fachmedien Wiesbaden, 2018, pp. 469–480.

[19] S. Liban, “Decentralization: A trend of the digital revolution?,” March 6th, 2018. [Online] Available: https://medium.com/@SamLiban/decentralization-a-trend-of-the-digital-revolution-af1c64806446. Accessed on: July 13th, 2018.

[20] C. W. Ibbs, C. K. Wong, and Y. H. Kwak, “Project Change Management System,” *Journal of Management in Engineering*, vol. 17, no. 3, pp. 159–165, 2001.

[21] J. D. Herbsleb and D. Moitra, “Global software development,” *IEEE Softw.*, vol. 18, no. 2, pp. 16–20, 2001.

[22] V. Venkatesh and H. Bala, “Technology Acceptance Model 3 and a Research Agenda on Interventions,” *Decision Sciences*, vol. 39, no. 2, pp. 273–315, 2008.

[23] L. Jones *et al.,* “Employee perceptions of organizational change: Impact of hierarchical level,” *Leadership & Org Development J*, vol. 29, no. 4, pp. 294–316, 2008.

[24] M. Jennex and L. Olfman, “Assessing Knowledge Management Success,” *International Journal of Knowledge Management*, vol. 1, no. 2, pp. 33–49, 2005.

[25] A. Cockburn and J. Highsmith, “Agile software development, the people factor,” *Computer*, vol. 34, no. 11, pp. 131–133, 2001.

[26] K. Nyborg, “Project Evaluation and Decision Processes,” *Discussion Papers Statistics Norway*, vol. 1995, no. 137, 1995.

[27] N. Pollock, R. Williams, and R. Procter, “Fitting Standard Software Packages to Non-standard Organizations: The ‘Biography’ of an Enterprise-wide System,” *Technology Analysis & Strategic Management*, vol. 15, no. 3, pp. 317–332, 2003.

[28] O. N. D, M. Salleh, and Noorminshah, “Knowledge Sharing in Workplace: Motivators and Demotivators,” *IJMIT*, vol. 3, no. 4, pp. 71–84, 2011.

[29] R. K. Smollan, “Causes of stress before, during and after organizational change: A qualitative study,” *Journal of OrgChange Mgmt*, vol. 28, no. 2, pp. 301–314, 2015.

[30] J. Hassard, K. R. H. Teoh, G. Visockaite, P. Dewe, and T. Cox, “The cost of work-related stress to society: A systematic review,” (eng), *Journal of occupational health psychology*, vol. 23, no. 1, pp. 1–17, 2018.

[31] D. Yakimovich, J. M. Bieman, and V. R. Basili, “Software architecture classification for estimating the cost of COTS integration,” in *Proceedings of the 21st international conference on Software engineering - ICSE '99*, Los Angeles, California, United States, 1999, pp. 296–302.

[32] Microsoft, “Microsoft to acquire GitHub® for $7.5 billion,” Redmond, Washington., June 4th, 2018. [Online] Available: https://news.microsoft.com/2018/06/04/microsoft-to-acquire-GitHub®-for-7-5-billion/. Accessed on: July 13th, 2018.

[33] GitHub®, “Tracking the progress of your work with project boards,” [Online] Available: https://help.GitHub®.com/articles/tracking-the-progress-of-your-work-with-project-boards/. Accessed on: July 13th, 2018.

[34] Kanbanize, “What is Kanban,” [Online] Available: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban/. Accessed on: July 13th, 2018.

[35] N. A. Batista, M. A. Brandão, G. B. Alves, A. P. C. da Silva, and M. M. Moro, “Collaboration strength metrics and analyses on GitHub®,” in *Proceedings of the International Conference on Web Intelligence - WI '17*, Leipzig, Germany, 2017, pp. 170–178.

[36] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, “Social coding in GitHub®,” in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*, Seattle, Washington, USA, 2012, p. 1277.

[37] H. Domanski, “Microsoft acquires GitHub®, the world’s largest source code platform: Software giant scoops up more collaborative coding,” June 4th, 2018. [Online] Available: https://www.techradar.com/news/microsoft-reportedly-acquires-GitHub®-the-worlds-largest-source-code-platform. Accessed on: July 13th, 2018.

[38] T. Ohno, “TOYOTA TRADITIONS: "If a problem is left unsolved and the supervisor in uninformed, neither kaizen nor cost reduction can be applied. When there is trouble, stopping the machine means also identifying the problem. Once the problem is clear, kaizen becomes possible.",” Mar. 2004. [Online] Available: http://www.toyota-global.com/company/toyota_traditions/quality/mar_apr_2004.html. Accessed on: July 13th, 2018.