MASTER

On not sharing an office
communication tools in the age of virtual teams

van Dalen, T.M.

Award date:
2016

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
On Not Sharing an Office

*Communication Tools in the Age of Virtual Teams*

Tim van Dalen

Committee:
dr. Alexander Serebrenik
dr. ing. Anton Wijs
dr. Dirk Fahland

Supervisors:
dr. Alexander Serebrenik
drs. Jelle Oskam (Nubis)

August 31, 2016
Abstract

Software development teams consisting of developers that all work from different physical locations, known as virtual teams, are becoming increasingly more common. These teams have similarities to both open teams and classic development teams where everyone is in the same office, but due to the way that virtual teams are organized, there are also differences to both.

We looked at how these similarities and differences affect the way that developers in these teams make use of communication tools. While developers in virtual teams use communication tools for the same team-related purposes as developers in open teams, developers in open teams use communication tools for a broader set of purposes.

Furthermore, we looked at how the indicators for performance in virtual teams differ from those for open teams. We see that due to the extra organizational structure virtual teams have higher level measures available, but that number of commits, which is used a lot in earlier work on open teams, is a viable measure for virtual teams as well.

Then, we looked at how the use of communication tools in virtual teams affects the actual work to see if removing bottlenecks in communication tools would be a viable option for improving the software development process. We have found that communication activities accelerate work activities and that working and chatting are inseparable in virtual teams.

Finally, we have identified problems in the communication tools and ability to measure productivity in virtual teams and propose solutions to solve these problems.

We have shown that it is possible to apply communication tool improvements developed for open teams to virtual teams and that communication activities accelerate work activities in virtual teams. We conclude that improving communication tools will have not only a positive effect on the satisfaction of a virtual development team, but is in fact a requirement for reaching the maximum possible productivity of a team.
Acknowledgements

I would first like to thank my committee members: dr. Alexander Serebrenik, dr. ing. Anton Wijs and dr. Dirk Fahland. I want to especially express my gratitude to my supervisor, Alexander, for this most welcome suggestions and helpful feedback throughout this project. Special thanks to Alexey Zagalsky and Bin Lin for sharing in review and original versions of their work with me. Finally, I would like to thank Nubis and my supervisors at Nubis, drs. Jelle Oskam and Duco ter Steege for providing me with a place to perform my research and access to their resources.
List of Figures

2.1 Percentage of remote jobs in the monthly Who is hiring? threads on Hacker News ............................... 12

3.1 Screenshot from our survey on TypeForm ...................... 18
3.2 Survey completion times, with a median of thirteen and a half minutes ........................................... 19

5.1 Organization webhook settings for GitHub ....................... 36
5.2 Events from the GitHub data source (commits) and the Hangouts data source (sent messages) ......................... 43
5.3 Events from the GitHub data source (commits) and the Hangouts data source (sent messages) for a week .................. 43
5.4 Event time versus event index for Hangouts message events of one developer ........................................ 45
5.5 Event time versus event index for GitHub commit events of one developer ............................................ 45
5.6 The box-and-whisker diagrams for real and simulated evaluation and response latencies ................................. 47

6.1 Waffle project board .................................................. 55
6.2 Hangouts chat showing the Nacho github plugin ................ 56
6.3 Dashboard view of verm, with as main plots the hours worked by project and developer in the last 30 days ............... 58
6.4 Viewing a single developer on verm ............................... 59
6.5 Viewing a project in hub ............................................. 61
6.6 Registering a new recruit in nova ................................. 62
6.7 Viewing a recruit in nova ............................................. 62

A.1 Submission dates for the survey results submitted after the first email. Role ‘None’ means that the respondent does not have a position that is directly related to the software development process ............................................ 73
A.2 Submission dates for the survey results submitted after the first reminder for everyone ................................... 74
A.3 Submission dates for the survey results submitted after the reminder for developers ........................................ 74

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>Graphs for developer D1</td>
<td>81</td>
</tr>
<tr>
<td>C.2</td>
<td>Graphs for developer D2</td>
<td>81</td>
</tr>
<tr>
<td>C.3</td>
<td>Graphs for developer D3</td>
<td>81</td>
</tr>
<tr>
<td>C.4</td>
<td>Graphs for developer D4</td>
<td>82</td>
</tr>
<tr>
<td>C.5</td>
<td>Graphs for developer D5</td>
<td>82</td>
</tr>
<tr>
<td>C.6</td>
<td>Graphs for developer D6</td>
<td>82</td>
</tr>
<tr>
<td>C.7</td>
<td>Graphs for developer D7</td>
<td>83</td>
</tr>
<tr>
<td>C.8</td>
<td>Graphs for developer D8</td>
<td>83</td>
</tr>
<tr>
<td>C.9</td>
<td>Graphs for developer D9</td>
<td>83</td>
</tr>
<tr>
<td>C.10</td>
<td>Graphs for developer D10</td>
<td>84</td>
</tr>
</tbody>
</table>
## List of Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Script used to get the timestamps for all sent Hangouts messages</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>Ruby script to bin timed events into minute window intervals</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>R script used to perform the Granger tests</td>
<td>39</td>
</tr>
<tr>
<td>5.4</td>
<td>Ruby script to generate simulated in communication events</td>
<td>41</td>
</tr>
<tr>
<td>5.5</td>
<td>Ruby script to calculate $\tau_E$ and $\tau_R$</td>
<td>42</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In past years, software development teams consisting of developers that work from different places have become increasingly common. This can be attributed to a multitude of reasons. For instance, with better internet connections becoming available, it is finally practical to work this way. The shortage of developers in Silicon Valley also causes companies to hire developers in remote locations \[20\]. Furthermore, we know from personal experience that developers simply like working from home.

In this thesis, we look at how these (virtual) teams use communication tools to get their work done and how their use of communication tools differs from that of other types of teams.

Our aim in this is to identify bottlenecks in the use of the tools that these teams use. Since virtual teams work similarly to open teams, we expect them to use communication tools in a similar way and experience the same bottlenecks. If the bottlenecks for virtual teams are the same as those for other types of teams, we might be able to re-use earlier results to improve communication tools for virtual teams. If not, we believe the differences we find will provide a starting point for developing better tools for virtual teams.

This thesis is organized in the following way.

First, in Chapter 2, we look at the different types of software development teams that exist and further explore the definition of virtual teams. Then, we formulate our problem statement and discuss the virtual team we will study in this thesis.

To accomplish our aim, we ask four different research questions.

In Chapter 3, we focus on the differences between how virtual and open teams use communication tools to get their work done with RQ1: What
are the differences in usage of communication tools between virtual teams and open teams?

Then, in Chapter 4 we formulate what it means for a virtual team to get 'work done' by talking to different stakeholders within the team we study using RQ2: What are key performance indicators for virtual software development teams in the industry?

Next, we combine the results of the previous two chapters to study the impact of communication tool use on work in Chapter 5, where we use RQ3: How does the use of communication tools impact these performance indicators?

In Chapter 6, we ask ourselves RQ4: How can we improve communication tools for virtual teams and how can we measure their impact? We identify problems with communication tools and productivity measures within the team we study and propose solutions for them.

Finally, in Chapter 7, we summarize our results from the previous chapters and discuss the work that be done to further research into this subject.
Chapter 2

Virtual Software Development Teams

2.1 Types of teams

Since most software development projects are large, work usually happens in teams of multiple developers.

Mantle and Lichty identify five different types of programmer-company relationships, based on their proximity to the organization [21, p. 80-88]. These types are in-house employees, geographically distant employees, contractors, contracted managed teams and outsourcing companies. In the last two cases, this proximity is based on the customer organization, i.e. the one hiring these outside teams. In-house employees and geographically distance employees are both direct employees of the company, while a contractor is an outsider hired to complete a certain task. For our purposes, the last three types are equivalent since they result in the same kind of team, albeit with different management structures.

Historically, most software development has happened within teams of in-house employees. We call these types of teams ‘classic’ teams.

If a team consists of contractors, contracted managed teams or outsourcing companies, we consider this to be a ‘freelance’ team. Of course, individual contractors work differently from entire contracted teams, but their relationship to the client company is very similar. Because of this, we group them here.

Teams that consist of geographically distant employees are considered ‘virtual’ teams. If all team members are in the same location but the entire team is distant (due to outsourcing or off-shoring) we consider the team a freelance team.
Sometimes teams can be made up of developers with different types of programmer-company relationships. For instance, it is not uncommon for classic teams to be supplemented with some geographically distant employees or contractors. In those cases, we use the relationship type that is in the majority for the team classification.

2.1.1 Classic team

Within a classic team there is a clear employer-employee relationship, along with a (usually) well-defined hierarchy. Software developers working in classic teams usually get their work directly assigned to them. Communication within classic teams mostly happens face to face, since this is simply the easiest way to have a discussion with someone close to you [13].

2.1.2 Freelance team

In contrast to developers working in classic teams, freelance developers do not have the traditional employee relationship, but rather a client relationship with the company that hires them. This means that companies typically should not treat them as employees [21] with regards to benefits. Since they are typically hired on a task by task basis, freelance developers have more freedom to choose which type of projects they want to work on. In effect, they are their own boss, which can cause the relationship between a team manager and team member to be drastically different.

Within freelance teams conflicts of interest can arise when a freelancer is hired to complete a task and not for a certain time frame. In such a situation it is in the best interest of the developer to finish the task as quickly as possible, which can result in them hiding technical debt. The fact that the interests of team members (and managers) can diverge to this degree (in a normal, healthy situation) is unique to freelance teams.

We call a team that consists of (mostly) freelance developers a freelance team.

2.1.3 Open team

There are many different ways to define an open team. Here, we call any team that practices software development in the open (so for which communication is public) an open team.

Team dynamics and relationships within open teams can vary greatly. Open teams can consist of volunteers that work on the project in their spare time. In this case, there can be one person who will assume the role of leader (also known as benevolent dictator).
In the case of volunteers, it is harder to enforce anything since there are no formal reward structures or arrangements in place. While it is possible to reward or punish people through social mechanisms, this increases the risk of people getting their feelings hurt and ultimately quitting the team.

Sometimes companies hire employees to work on or lead an open team. In this case the company can steer the direction of the project and there is some way to enforce who works on what and when.

Since, by definition, most of the software development process of open teams happens in public, this type of team is the easiest to look at from the outside. Thus, a lot of research into how these teams work has been done [19, 27, 32].

2.2 What makes a team virtual?

A virtual software development team is a team that consists of professional developers that are not in the same physical location.

In its way, it is not a completely new type of team, but rather a mixture of the team structures presented earlier. In a virtual team, all developers are generally employees of the company they work for, as opposed to the ‘every man for himself’ freelance structure. However, compared to developers working in classic teams, they usually have much more flexibility with regards to working hours. Virtual teams usually focus more on the work done than the hours worked - as long as deadlines are met it is less important when the work gets done.

Since virtual teams lack the ability of face-to-face communication but do share everything as one big team (as opposed to freelancers who might have their own agendas) they can typically use the same communication tools as open teams. This also makes their behaviour easier to study than the behaviour of classic teams. As long as we can get access to the event archives of all communication tools a virtual team uses, we can analyze them as well as open teams.

The biggest differences between open teams and virtual teams stem from the fact that developers that work in a virtual team are being paid to do so. This obviously affects the power of an individual developer within the team. Developers in a virtual team have a ‘boss’ that determines what they work on, whereas developers in open teams are usually only driven by either their own motivation or the fact that people are depending on them [12, 25, 29, p. 108-111]. A lot has been written about fatigue and burn-out in the open source community [2, 5, 6, 18]. Since developers working in virtual teams are actually ‘working’ professionally (and thus have more
clearly defined borders of work), it can be easier for them to keep a healthy work-life balance but having the freedom of deciding your own hours can make this more challenging for some people [14].

For several reasons, virtual teams are becoming more widely implemented within the software development industry.

First, we know from personal experience and from talking to other developers that developers simply like working from home. Furthermore, a shortage of developers in Silicon Valley is accelerating the demand for developers to move there, or, if that is not feasible, to work remotely [20]. Also, technological progress is an important driver in actually making it possible to work this way [28].

Figure 2.1, taken from the blog post by by Sebastian Pawlus[1], shows the percentage of remote jobs advertised in the monthly Who is hiring? threads on Hacker News, a popular news aggregator and discussion platform for software developers. In these discussion threads, companies list their open positions so potential candidates can read them. The word remote is used to indicate positions within virtual teams on a permanent (non-freelance) basis.

This brings us to the situation we are in now: more and more developers are working within virtual teams, but we currently do not have very good insight in how these types of teams work.

---

2.3 Communication as a part of the software development process in virtual teams

Due to the way virtual teams work, we suspect that improving communication tools will have a positive impact on their productivity.

Because of this, we want to find out how virtual software development teams in the industry use communication tools, and how this affects their work. To that end, we ask the following questions:

**RQ1:** What are the differences in usage of communication tools between virtual teams and open teams? We ask this question to find out if we can re-use results obtained for open teams for virtual teams later on. We discuss this research question in Chapter 3.

**RQ2:** What are key performance indicators for virtual software development teams in the industry? In Chapter 4 we look at how the performance indicators for virtual development teams differs from those for open development teams so we can compare their output for RQ3.

**RQ3:** How does the use of communication tools impact these performance indicators? Using the results from RQ1 and RQ2, we look at the relation between communication and work activities in Chapter 5.

**RQ4:** How can we improve communication tools for virtual teams and how can we measure their impact? With the results obtained for RQ3, we look at the specific problems that exist for the Nubis team and try to solve these by developing new tools.

To help us answer these questions, we embed ourselves in a virtual development team within the industry and look at their data as well as talk to their developers.

2.4 The Nubis development team

Nubis is a full-service digital agency that handles everything from concepts to execution.

The web and application development team consists of fifteen people, of whom all but one work part-time and are in some way still students (ranging from bachelor students to PhD students). Everyone works from home or their university. Thus, the Nubis development team is a virtual team.

Most developers in the Nubis development team have a so-called ‘zero hours contract’ (nulurencontract). With this contract form an employee is an actual, direct, employee of the company, but there is no guarantee of a minimum number of hours of work each week. This works very well for students, who might have busy periods such as exam weeks during which
they will not be able to work. For the company, it means having the ability
to quickly scale down labour costs if there is not enough work.

This contract form emphasizes the flexibility of a virtual team. Without traditional working hours it is easy to remove the overhead of downtime and just work when there is actual work to be done. This also means that work can happen whenever it is desired by a client or whenever a developer would like - for instance during the night. This presents extra challenges for project managers, but also the opportunity to deliver within very short time frames. For a classic team, there is just one hour between 16:00 on a Friday afternoon and 9:00 on the following Monday. For a virtual team like the one at Nubis, there are 65.

The Nubis team also includes designers, project managers and online marketeers.

Part of the reason we suspect virtual and open teams to use communication tools in a similar way is due to the way they are organized. The team at Nubis is organized by mutual adjustment and a mix of standardization of skills and standardization of norms [8, p. 49-50] [22, p. 101]. In this respect, the Nubis development team is not unlike an open team [15].

Since Nubis developers work in virtual teams, all communication between developers and project managers happens online (either through Hangouts, GitHub or the occasional email). Usually, this communication is real-time, as all team members are within two time zones of each other and a lot of people are working at the same times. This means we can access logs of (almost) all discussions between developers, as opposed to discussions happening between developers within the same office. Occasionally developers will meet for a face-to-face whiteboard session which is not structurally logged.

For most (large, long-term) projects, a Hangouts group chat exists. This group chat includes a bot (Nacho), that has several plugins (including sending notifications of project board updates or failed continuous integrations). A plugin can either be triggered by a chat command (starting with a !) or an outside event (through the use of webhooks). Nacho was built by Nubis and is easily extensible.

Nubis and its developers have given us access to their team and have made a large amount of historical data available for us to analyze from the data sources below. Most of these sources offer event data. That is, each data point has a date and time and possibly metadata, but no value.

**Google Hangouts**
The team uses Hangouts as their main communication channel. This data
source has all messages sent and received to both private chats (one on one) and group chats, including formatting, emoji and attachments (such as images and locations).

**GitHub**
GitHub is used as a central point of truth for all projects. Aside from all commits made by all developers, we also get bug tracker data from this source. GitHub issues are labeled according to a scrum-like practice where a bug or feature moves through the project board. By looking at ‘labeled’ and ‘unlabeled’ events, we can derive when a certain feature was started on, finished and tested.

**CodeShip**
This service is used for continuous integration and deployment. It gives us information about successful and failed builds.

**TrackMyHours**
All team members use TrackMyHours to log their time. For each entry we have a timestamp, duration, project worked on and an optional description. Most team members ‘clock in’, giving us accurate timestamps, while others enter cumulative data on a weekly or monthly basis. Note that data points for this source are time spans rather than events.

We note that the author has been embedded in this team for more than five years already. Because of this, we will be able to give our own insights at times and use elements of ethnography in our research [16].
Chapter 3

Communication in open and virtual teams

In this chapter we look at **RQ1**: What are the differences in usage of communication tools between virtual teams and open teams?

3.1 Methodology

In order to answer this question, we must first get an overview of how our team uses communication tools. To get this overview, we conducted a survey in the team.

To get a broad overview of how team members see their use of communication tools, we have designed a survey. This survey is based on the work of Lin et al. in *Why Developers Are Slacking Off* [19]. The survey designed in *Why Developers Are Slacking Off* is specifically made for developers that use Slack as their primary means of communication. The questions aim to find out how developers are using Slack to get their work done. This aligns with our goal of wanting to find out how our team uses communication tools.

To design our survey, we started with the questions from the survey about Slack and modified them to make more sense for our team. For most questions, this meant simply replacing questions about Slack-specific features with questions about Hangouts features. We replaced questions about Slack bots or integrations with questions about Nacho. Since the use cases of Slack (and integrations) are so similar to those of Hangouts (and Nacho), in most cases this was possible without changing the meaning of the questions involved.

We replaced the demographics questions with some more specific ones that make more sense for our team.
While we do not perform an exact replication \[26\] of the earlier work, we chose to stay close to the survey as designed in *Why Developers Are Slacking Off* so we can compare the results of our survey on Hangouts usage within our virtual team to the results of the survey on Slack usage within open teams.

The survey was sent to everyone at Nubis, not just to people who are in the development team. Respondents were asked to describe their role in the software development process, so we can identify the different types of stakeholders in the development process.

The complete survey can be found in Appendix A.

The survey was sent to possible respondents via email. The first email was sent to all people in the Nubis domain with a short explanation of the goals of the survey and a link to learn more about this research. Upon opening the link to the survey, respondents are greeted with an introduction which further explains the goals of the survey and how the results will (and will not be) used. Direct results from the survey were not sent to management and were anonymous.

A second email was sent to everyone at Nubis to remind them a week later. This email included a deadline of four days later.

Later, the last reminder email was sent specifically to people in the development team, reminding them to fill in the survey. A few days after this email was sent we started analysis of the results.

The full text of all emails can be found in Appendix A.3.

The survey was sent using Typeform\(^1\). Typeform is a service that allows us to easily setup clean-looking web forms that behave well on both desktop and mobile browsers.

Typeform also offers us some rudimentary analytics, such as the average time it took respondents to complete the survey or the device they completed the survey on. Figure 3.1 shows what the survey looked like to respondents.

After completing all questions, the respondents saw a ‘thank you’ message with the promise of receiving the outcomes of the survey at a later time. We will discuss this further in Section 3.3.1.

### 3.2 Results

#### 3.2.1 Response rate

In total 28 people responded to the survey. The survey was sent out to 38 people, resulting in a response rate of 74%. The respondents are representa-

\(^1\)http://typeform.com
Figure 3.1: Screenshot from our survey on TypeForm
Within the development team, the response rate was a little higher at 87% (13 out of 15). The respondents are representative of the development team. This is likely due to the fact that we sent an extra email to all developers which resulted in two more responses from within the development team.

Most respondents completed the survey immediately after receiving an email about it. For a complete analysis of the response dates, see Appendix A.4.

### 3.2.2 Completion time

Figure 3.2 shows a boxplot of the survey completion times, with a median of 13.53 minutes. This corresponds to our estimation of the time it would take respondents to complete the survey (which was 10 to 15 minutes).
The outliers are probably due to people who opened the survey, then went to work on something else, and completed the survey later. Because of this, we choose the median to represent completion time instead of the arithmetic mean.

Ten people opened the survey on their mobile phones after following the link from the email, while only three people actually completed the survey on their phone. Average completion time on mobile phones is a lot higher than on desktop and laptop devices.

### 3.2.3 Open questions

In this section, we look at the answers to the open questions in our survey.

**How often do you use hangouts for working?**

This question was overwhelmingly answered with ‘most of the workdays’. Only one person indicated that they do not use Hangouts for working. That person is not in a role that has any direct connection to the software development process.

**What do you use Hangouts for?**

To show the purposes respondents to our survey use Hangouts for, we coded the responses to this question [31, p. 804]. The results of this coding can be found in Table 3.1.

**Benefits from using Hangouts**

Below are the most frequent answers respondents to the survey gave to the question ‘What benefits do you get by using Hangouts?’.

- It’s fast
- There’s a low threshold to reach people
- It works on multiple platforms (there is a desktop application and a mobile application)
- All work related things in one platform
- Less email traffic
- Being able to reach someone directly
- Audio/video conferencing
- It feels professional

---

20
Note that most of the answers given here are not specific to Hangouts, but are instead properties of instant messaging platforms in general.

Some specific downsides of using WhatsApp instead of Hangouts were named: it’s not professional and the desktop app doesn’t work very well. One Nacho advantage was named: being able to see who is working on what.

**Frustrations**

Below are the most frequent answers respondents gave to the question 'What do you get frustrated with while using Hangouts?'.

- People not being online or not responding
- Switching between multiple Google accounts
- Having a single timeline in a chat
- Multiple image upload is not working\(^2\)
- Only being able to upload images\(^3\)
- Getting lots of notifications in group chats while off the clock
- Can not upload images by pasting them in the chat field
- Synchronization between desktop and phone is sometimes ‘off’
- Having to turn off notifications when unavailable
- Ugly emoji
- Not being able to add shared tasks or notes
- It looks too casual

The following frustrations respondents submitted are specific to Nacho and not Hangouts.

- Getting a notification when someone in the team made a commit that wasn’t linked to an issue (See Section 6.2.2)
- When a plugin does not work well
- Experiencing a Nacho crash

\(^2\) This is a feature that was working in the past and has been removed from Hangouts
\(^3\) Again, this was available in the past
Possible improvements

Below are suggestions survey respondents submitted on how to improve Hangouts.

- Better notification control
  - Mute individuals in a group chat
  - Temporarily auto-mute a chat when it’s minimized instead of popping back up with every message
- Better image viewer
- Being able to ping someone
- Make it easier to switch between accounts
- Replying per message instead of per chat
- Better emoji designs
- Being able to favorite messages so you can find them later
- Visual update
- Add a search feature in the chat window

Possible improvements to Nacho/plugins

- Better notification control
- Help function with overview of all possible commands
- Being able to get an on-demand summary of actions of the previous day in a specific project

Possible new plugins

Below are some of the suggestions team members have for new plugins.

- Connect to the invoice system
- Add a reminder function
- Add a polling function

---

4It should be noted that this would be a fundamental change to the platform. Having threaded replies is usually not a feature found in instant messaging platforms. It is a feature more commonly associated with discussion platforms which have less focus on being real-time.
• Schedule community managers for the next month based on availability
• SSH client
• RDP client
• Drawing on screenshots
• See who is clocked in on a project
• Calendars
• Shared notes
• Log hours using a command

**Other channels**

One respondent uses Slack for personal purposes and would like to try it out for software development. One person uses Facebook for software development purposes.

### 3.3 Discussion

From our survey we can see that almost everyone on the team exclusively uses Hangouts for work-related chats. This means it will be a good data source to perform analysis on in later chapters.

We notice a lot of opportunities to improve user satisfaction in Hangouts usage. We recommend Nubis to explore other instant messaging platforms or - if possible - develop their own Hangouts client.

Now that we have gained understanding into how our team uses communication tools, the next step is to compare these results with those of open teams. We do this by looking at two earlier studies.

**Why Developers Are Slacking Off**

The first of these earlier studies, *Why Developers Are Slacking Off* [19], was sent to 30 open teams related to software development. Since our questions are largely based on the questions posed in this study, we were able to compare answers on a per-question basis.

Table 3.1 shows a comparison of the use cases for those developers using Slack and our developers using Hangouts. The biggest difference we note between the studies is that developers in the Slack study also use Slack for participation in communities of practice, while people in the Nubis team
Purposes | Our study
--- | ---
**Personal**
Discovery and news/information aggregation | 12 | 0
Networking and social activities | 5 | 2
Fun | 8 | 2

**Team-wide**
Communication | 77 | 25
Team collaboration | 17 | 23
Customer support | 4 | 0
Dev-ops | 23 | 2

**Community-wide**
Participation in communities of practice | 22 | 0

Table 3.1: Survey results of what developers use Slack for (104 responses) versus what our team uses Hangouts for (28 responses)

do not. This could be explained by the lack of Nacho plugins available for these purposes. However, none of the responses to the question ‘If anything is possible, what kind of plugins would you like to have beyond the existing ones?’ fall in that category, so this might just be something our team is not interested in. Aside from that, the distribution of purposes is similar. The higher number of respondents in the Slack survey that use Slack for ‘Discovery and news/information aggregation’ and ‘Dev-ops’ purposes might be explained by the fact that a lot of bots and integrations exist for these categories, while only one basic Nacho plugin (notification of deployed builds, dev-ops) exists.

Social and Communication Channels in Software Development

In *How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development* [27], Storey et al. identify the types of communication channels used by software developers, and key challenges developers face with communication challenges.

We compared the results of the open-ended questions of our survey with their results.

Storey et al. remark that ‘Distractions and interruptions from communication channels negatively impact developer productivity’ [27]. This is a sentiment echoed in the responses we received (I get frustrated by... ‘[notifications about] unreachable commit[s]’, ‘not being able to effectively mute a group chat’, ‘messages of unimportant stuff in group chats’) to the questions about frustrations with using Hangouts and Nacho. Storey et al. also note that miscommunication on text-based channels is common and that people
are challenging regardless of the channel used. This is something we also see in the results of our survey. While people have been polite about the subject of miscommunication, some people remark that it can be challenging when ‘you do not get a response’, ‘not everyone is used to use it’ (which might be solved by Nubis with more training) or ‘when people are not online’. This indicates that there is a feeling that people should always be online, even though this requirement itself is perceived as a negative pressure.

We have learned that our team uses Hangouts for most intrateam communication. Based on the comparison of our results with other studies on open teams, we conclude that open and virtual teams use communication tools in a similar way for team related purposes. We also saw that they face the same kind of challenges using communication tools. We note that we saw indications that open teams might use communication tools for personal purposes more frequently than virtual teams, although further research into this will be required.

3.3.1 Evaluation

We presented our results from the survey as part of a set of internal talks at Nubis. Around 15 of the respondents to our survey were present, most of them from the development team subset of the respondents.

During a discussion, most attendees agreed with the results we presented. In particular, attendees recognized the most common frustrations we presented. Nubis management noted that a lot of the frustrations team members mentioned are Hangouts-specific, while the benefits team members mentioned are general to instant messaging platforms. Nubis will look into switching to a different platform or solving problems with Hangouts using extra tooling.

3.4 Threats to validity

In this section, we identify possible threats to the validity of the results we obtained above. We can identify three types of threats [23].

Construct validity deals with the validity of the methods used in the experiment.

Internal validity is related to the execution of the experiment and making sure that no other outside influences (other than the ones under investigation) impact the results.

In this chapter, we identify a threat to internal validity. By sending an extra survey reminder to the people in the Nubis development team (as
opposed to those working at Nubis but not in the development team), we have introduced a selection bias towards the development team. This means we cannot compare the results from the development team with results from outside of the development team.

Another threat to internal validity we note is caused by the relative lack of Nacho plugins, in comparison to the number of Slack bots and integrations available. The different use cases of our team members compared to respondents in the Slack survey might be explained by the fact that some of the things they might want to do are simply not currently possible. We note that we asked respondents what else they would like to see in plugins, but we recognize that respondents might not be aware of what they are missing.

**External validity** means that we can generalize the results from the experiment to other settings.

We also identify a threat to external validity. In order to generalize our results to all virtual teams, the development team at Nubis must be a typical virtual team.

In this chapter, we have presented a survey we conducted with the Nubis team. We have compared the results from this survey with another survey conducted on open teams and concluded that the communication tool use of virtual teams is similar to that of open teams, though open teams may have a broader set of use cases. We will use these results in Chapter 5 when comparing the communication and work activities of open and virtual teams.
Chapter 4

Measuring output of a virtual team

In this chapter we look at RQ2: What are key performance indicators for virtual software development teams in the industry?

4.1 Methodology

In order for us to be able to talk about the influence of communication tool use on the software development work itself, we must first define metrics by which we judge the ‘output’ of the software development process. Finding suitable metrics for this purpose has historically been a hard problem, for which there exists a lot of previous work [3,4,10,24].

Since we are interested in the relationship between communication and these metrics, and not which metric is the best tool for this job, we simply asked stakeholders in the development process at Nubis what metrics they use.

To this end, we have conducted two semi-structured interviews. Duco ter Steege, Partner and day-to-day manager at Nubis, offers a look at this problem from a management perspective. Project manager Philip Vermeij handles large projects and is responsible for making sure deadlines are met and customers are kept up to date. His view concerns daily output and making predictions for future output.

Since the author is also a stakeholder in the software development process at Nubis, we will also give our opinion on this problem.

In the results from these interviews, we will identify a few key concerns that we will use in Chapter 6 to motivate problems.
4.2 Results

The full transcripts for the interviews we have conducted can be found in Appendix B. Below, we list the key insights from these interviews.

4.2.1 Philip Vermeij

Philip’s main role in the software development process is project manager. Since he works regular office hours while most of the developers do not, he has two main different strategies for measuring performance: when checking in in the morning and during the day.

When he arrives in the morning and wants to check in, he gets most of the relevant information from the email notifications about GitHub issues that he has received during the night. Philip goes through these notifications one by one to see which issues have gotten attention since he last worked and to see if he needs to take any action. He also looks at the issues that he has not received any notifications about by comparing to his own ‘shortlist’ of important issues.

After he has performed this check in in the morning, Philip instead tries to follow work activity live. He does this by using the github Nacho plugin

In both of these cases, Philip eventually ends up in issues. For him, issues are the place where productivity happens. Within issues, he looks at both the commits made and the commit status updates performed by quality assurance engineers and developers.

Furthermore, Philip uses the online status of developers to gauge their availability (concern I2). When he needs someone quickly for a time-sensitive issue, he will usually go to developers he knows are online a lot.

4.2.2 Duco ter Steege

For Duco, getting an overview of productivity means doing this at a much higher level. He looks at the money in and out flows for each department, including the development team.

His main indicator of productivity is hours worked, which he tries to keep balanced with hours sold. He needs to have a good overview of both the labour costs for a given period of time and the invoices that were sent in that same period (concern I3).

Duco is also interested in the work-life balance of Nubis employees and the (perceived) pressures that a flexible contract can bring. He wants to see the number of hours employees log on an average day and conduct talks
with people that are logging too little (concern I4) or too much (concern I5) time.

4.2.3 Tim van Dalen

Tim has ultimate technical responsibility for the software developed at Nubis. For Tim, productivity is measured both by the code committed and (to a lesser extent than Philip) the features these commits implement.

To see if a certain project is getting enough attention, Tim looks at the activity in the Hangouts group chat for the project. There, he uses the amount of messages sent by Nacho as a gauge for the productivity level (concern I6).

At a whole team level, Tim looks at the organization activity feed in GitHub to see all commits that were made recently. This, in combination with the messages for these commits and the variation in developers that made them are his indicators for productivity of the team.

Tim would like to have a better way to see the number of hours worked by developers (and on which project) as the current export from the time tracking system is very limited (concern I7).

4.3 Discussion

We see that for Philip the main starting place for discovering productivity is the issue tracker. However, even within the issue tracker, productivity is usually still measured by the commits associated with the issue made.

For Duco, productivity is measured by the budgets for each running project.

For Tim, productivity is measured by commits and features implemented in them.

In research about open teams, commits are used to measure productivity as well. For instance, in [30], the authors use GitHub commits as a measure of personal productivity. In [32], which we will discuss in more detail in the next chapter, the authors use commits as reflection of work as well.

We identify the following performance indicators for virtual teams, from high-level to low-level:

- Budgets
- Issues in the tracker
- Commits
Based on the interviews we have conducted and our experience from being embedded in the Nubis development team, we conclude that, even though there are other indicators as well, the number of commits made is an important indicator for virtual teams, as it is for open teams.

4.4 Threats to validity

For this chapter, we identify the same external threat to validity as for the previous chapter. If we want to generalize the results obtained with regards to productivity measures from Nubis to all virtual teams, Nubis must be a typical virtual team.

In this chapter, we have seen that there are several performance indicators for virtual teams. In the next chapter, Chapter 5, we will use these performance indicators to build event logs of work activities and compare them with communication activities.
Chapter 5

Communication and work in a virtual team

In this chapter we look at RQ3: How does the use of communication tools impact these performance indicators?

5.1 Methodology

To answer this question, we looked at the relationship between communication events and the performance indicators we have found in the previous chapter.

Specifically, we have performed two experiments. In the first experiment, we use the Granger causality test [11] to see if we can find a relationship between chatting and committing. The second is a replication of the Time-series based method for measuring the interaction between communication and committing activities in [32].

5.1.1 Data collection

In our experiments, we look at two sets of events, one for communication and one for work activities. For communication, we use messages sent on Hangouts. For work activities, we use commits made.

In the previous chapter we found more complex types of events for work activities (such as those relating to issues) but since those still largely boil down to commits made and the study we want to compare our results with also uses commits as the source of work events, we choose to use commits here.

Hangouts messages

Google does not provide a simple way to find the timestamps for all messages that were sent by a user.
One approach is to monitor all communication with the Hangouts API and parse the protobuf\textsuperscript{1} encoded messages ourselves. However, the distributed nature of the team makes this infeasible.

Another approach is to have our bot, Nacho (as described in Section 6.2.2), save records of all communication and keep a log of all messages sent by developers. While this would work, it is easy to miss a lot of messages that might be sent in channels where Nacho is not added, such as private communications between two developers. Since these types of chats are potentially very relevant to work (for instance, developer D1 could ask developer D2 about an API that was built by D2, skipping the project chat entirely since D1 knows D2 built the API and is usually online) we want to make sure the approach we choose can capture them as well.

The last approach we tried before settling on our final one was to use the chat archive features in Gmail. An archive of all chats is saved to a special folder in Gmail, where it is searchable through the Gmail web interface. This special folder is also available through IMAP, allowing us to download a user’s chat history in MBOX format. Unfortunately, while this approach does give us the messages, the MBOX format is very limited and does not include attachments to messages (such as images and shared locations). Formatting is also very limited and a lot of emoji data is lost. Because of this, we abandoned this approach.

Through Google’s Takeout\textsuperscript{2}, Google allows users to download dumps of their data for a number of their products, including Hangouts. A user’s entire chat history is actually divided between two products; data from before Hangouts was launched (May 15, 2013) is available as email message chains through the Gmail dump, data since Hangouts was launched is available as a rich JSON file. Since the earlier history is in the same, very limited, MBOX format and we have a lot of data (since 2013) available in the much more interesting Hangouts dump, we discard it.

The JSON file is a dump of the user’s entire chat history. Most of contents is in an array of conversations (which includes both group and one-to-one conversations). A conversation has a number of participants and a number of events. Events can be title changes, people joining, people leaving, a video/audio chat starting or a chat message being sent. We are mostly interested in the last type of events.

There are some libraries available that can parse the Takeout dump into something we can use for our analysis, but none worked very well. Thus, we decided to develop our own. Due to the sheer size of the data (the single JSON file in the author’s dump is around 1.2 gigabyte), the standard

\textsuperscript{1}https://developers.google.com/protocol-buffers/docs/overview
\textsuperscript{2}https://takeout.google.com/settings/takeout
libraries of some languages were not able to reliably parse it, even when given large amounts of memory. The size of the dump also made it hard to explore the file by hand. Using a dump of a newly created account made this a bit easier. However, the data for that account was of course much less complete. Luckily, within a conversation, the schema is the based on the protocol buffer schema used by the Hangouts API, with which we are already familiar through the development of Nacho.

Using this knowledge, we developed a Ruby library that can load in the dump object and read it into some more useful classes that we can run our analysis on. The resulting library is available on GitHub and through Rubygems under the MIT license. Usage examples are given in the README.

We asked all developers in the team to request a Google Takeout archive of their Hangouts data and send it to us. Before starting the analysis, we first discard all one-on-one conversations where the other participant is not part of the company domain since some respondents to our survey said that they use Hangouts for personal communication as well.

Listing 5.1 shows the script we used to find the timestamps for one specific developer (it should be noted that the Google user id for this developer must be manually found and set). The file `nubis_ids.json` contains the Google user ids for all people in the domain, which we use to filter out direct messages with people that are not in the domain. Note that this does not filter out group chats that exist purely for personal uses. In our case, these did not occur.

Unfortunately, Google does not offer an easy way to get all Google user ids in a Google Apps for Business domain. Google does offer a Directory API as part of its Admin SDK but that was needlessly complex. When applying this method to a much larger domain, we recommend using this API.

We ended up joining the sets of users we discovered in all Hangouts exports and comparing the name for each user with the list of users in the domain (exported as CSV through the Domain Admin web interface). For duplicate names (which can happen when people chat with both company and personal Google accounts) we decided to keep both ids. From asking around within the company we discovered that sometimes people accidentally use Hangouts with their personal account to discuss work and that chats between company accounts and personal accounts (of employees) are generally work-related as well. Finally, we went through the list of names in

---

3https://github.com/nubisonline/hangouts_json_parser
4https://developers.google.com/admin-sdk/
Listing 5.1: Script used to get the timestamps for all sent Hangouts messages
the CSV export and checked if each name was present in the set of user ids at least once. For names that were not, we manually searched for the user’s company profile on Google+ and added the user id (visible in the URL of their profile) to the file.

The script works by loading in all data, performing some calculations and then dropping the user into an interactive shell. From there, the user can perform further analysis or save the results from the script to disk.

We executed this script for all developers in our team, saving the resulting timestamps as a comma separated values file.

**GitHub commits**

As with the Hangouts messages, there are multiple approaches to get the data we want from GitHub. For studies that focus on open teams, the easiest way to get data from GitHub is usually through projects that continuously collect GitHub data through the GitHub API and make it available as archives. However, since we are dealing with private industry data, that approach is not available to us.

An alternative is to scrape the GitHub API and go through all commits ever made on all repositories in the organization. The downside of that approach is that things can easily get messy when dealing with the GitHub API rate limits.

Luckily, there is an even easier way. GitHub offers the ability to set ‘webhooks’. A webhook is a simple API endpoint that is called whenever some event happens. In the case of GitHub, this can be a multitude of events, including one that is fired whenever a commit is pushed to a repository.

For organizations, GitHub allows administrators to setup organization-wide webhooks which are triggered for any event of the selected type(s) that occurs in the organization. This means we do not have to repeat the setup for newly created repositories, reducing the risk of missing data due to setup mistakes.

We created an endpoint within verm (see Section 6.2.2) to handle these incoming requests from GitHub and setup an organization webhook as shown in Figure 5.1. This setup was repeated for one organization, belonging to a client, that developers in our team frequently work in.

Whenever one or more commits are pushed to any repository within the organization, the webhook gets an HTTP POST request with metadata about each commit. Using this metadata, we look up the entity for the employee that made the commit and the entity for the project that the

---

5https://developer.github.com/v3/activity/events/types/
commit was made to. We then create a commit entity with the commit hash, timestamp and message and associate the employee and project.

While the project association is not directly used in this chapter, we use it within our verm dashboards. For instance, we show a comparison of commits and timelogs associated with a project, as well as a project ‘last active’ time.

This hook has been active since March of 2015, meaning that that is the start of our collected data from GitHub.

We then added a simple command to verm which, for each employee in the system, exports all timestamps that commits were made at as a csv file.

Events to time series

At this point we have, for each developer in our team, two files with timestamps of communication and work events. For the rest of the analysis however, it makes more sense to work with time series rather than sets of events.

Because of this we bin the events with windows of one minute. So, if within a minute three events happen, those values are replaced with a timestamp of that minute with value 3. If no events happen within a minute, the timestamp for that minute is added to the data with value 0.
We chose a window size of one minute after some experimentation. Sizes longer than a minute make it hard to see what was actually going on at a certain moment, with the risk of putting so much data within the same bin that relations are inevitable. Going with a window size of less than a minute makes the time series less useful. For instance, at a window size of a second the time series basically becomes the event set but with 0 values added for seconds without events. Since the timestamps we are using are generated by different servers (Google and GitHub) we were also cautious for small differences between the two data sources (which could be mitigated by binning with larger values) although our experiments indicated this was not a problem.

To perform this binning, we wrote a small script. It should be noted that the implementation is quite naive. However, since the size of our input data is manageable, we chose this approach because it is more intuitive and easier to understand. Listing 5.2 shows the script.

The script steps through every minute in the interval that we have data for and counts the number of events that happen within that minute, in both data sources. It exports this data individually, per data source, and as a merged table which has the number of events for both communication and work at one timestamp.

5.1.2 Experiments

Granger causality

To investigate the relationship between the communication event sets and the work event sets, we use the Granger causality test [11]. This test effectively tells us if one time series is a good forecaster for another.

Since Granger causality works on time series rather than sets of timed events, we use the time series we generated from the event sets earlier to carry out this test.

In order to use the Granger causality test, we must first make sure our data is stationary (meaning that the statistical properties of the series do not change over time). We do this by performing the augmented Dickey-Fuller test on our time series [7].

For each developer, we look at if the time series with communication activities is a forecaster for the time series with work activities.

Specifically, we use the `grangertest` function from R package `lmtest`, with order 1.

Listing 5.3 shows the R script we run for each developer. The null hypothesis of the Granger test is that the first time series does not help in forecasting the second time series.
require "rubygems"
require "active_support/core_ext/numeric/time"
require "date"
require "csv"

STEP = (1.to_f / 24 / 60)
DSTART = DateTime.parse(Time.at(1423361744).to_s)
DEND = DateTime.parse(Time.at(1470735036).to_s)

["D1", "D2", ...].each do |name|
  hangouts = CSV.table("#{name}-Hangouts.csv")[:time].sort
  github = CSV.table("#{name}-Github.csv")[:time].sort.map do |v|
    v - 21600
  end
  both_results = []

  DSTART.step(DEND, STEP).each do |date|
    next_date = date + STEP
    date_i = date.to_time.to_i
    next_date_i = next_date.to_time.to_i

    hangouts_count = hangouts.select do |v|
      v >= date_i and v < next_date_i
    end.count
    github_count = github.select do |v|
      v >= date_i and v < next_date_i
    end.count

    both_results << [date_i, github_count, hangouts_count]
  end

  CSV.open("#{name}-both.csv", 'w') do |csv_object|
    csv_object << ['date', 'github', 'hangouts']
    both_results.each do |row_array|
      csv_object << row_array
    end
  end
end

Listing 5.2: Ruby script to bin timed events into minute window intervals
library(ggplot2)
library(lmtest)

hangouts <- read.csv("hangouts_count_minute.csv", header = TRUE)
github <- read.csv("github_count_minute.csv", header = TRUE)
both <- read.csv("count_minute.csv", header = TRUE)

hangouts$date_i <- hangouts$date
github$date_i <- github$date
both$date_i <- both$date

github$date <- as.POSIXct(github$date, origin="1970-01-01")
hangouts$date <- as.POSIXct(hangouts$date, origin="1970-01-01")
both$date <- as.POSIXct(both$date, origin="1970-01-01")

ggplot(hangouts, aes(y=count, x=date)) + geom_line()
ggplot(github, aes(y=count, x=date)) + geom_line()

granger.test(github ~ hangouts, order=1, data=both)

Listing 5.3: R script used to perform the Granger tests

Working rhythms

In the second experiment, we perform a partial replication of Measuring the effect of social communications on individual working rhythms: A case study of open source software [32]. In particular, we look at the Time-series based method introduced in Section III.C of [32]. For the full methodology, we refer to that section. Here, we will give a brief overview.

The authors of the original work use in-communication for their communication event stream. In the previous experiment, we used sent chat messages. To create an in-communication event stream for use in this experiment, we use a modified version of Listing 5.2. Instead of only selecting messages where the sender id equals the id of the developer we are considering, this modified version only selects messages where the sender id does not equal the id of the developer.

The main idea behind this experiment is to generate a simulated set of events for communication, with similar properties to the original one, and comparing the way this simulated set interacts with the commit event set with how the real set of communication events interacts with the commit event set.
This interaction is measured by two properties: the evaluation latency ($\tau_E$) and the response latency ($\tau_R$). The evaluation latency is the time between a commit event and the next communication event. The idea behind this is that this next communication event could be someone else in the team responding to the commit. The response latency is the time between the last communication event before a commit and the commit. Here, the idea is that this time could be the time a developer takes to respond to evaluations and create a new commit based on them.

Since the authors of [32] did not publish any code for their experiments, we implemented the described algorithms in the paper ourselves. Listing 5.4 shows the script we used to generated simulated time series based on the in-communication time series. Figure 4 in the original paper describes how the simulated time series is built. In short, a list of $\Delta$’s is built for the times between each received chat message. Then, that list is shuffled and a new time series is built by applying the shuffled $\Delta$’s to the first time in the series one at a time.

Listing 5.5 shows our script for calculating the values for $\tau_E$ and $\tau_R$. The script shown is simplified for inclusion here. In reality, the script performs the calculations for the real and simulated communication sets at the same time.

Then, we make a box-and-whisker diagram with all values of $\tau_E$ and $\tau_R$ so we can compare the results for the real and simulated communication time series. If the values for the real communication time series are smaller than those for the simulated communication time series, this is an indication that communication events accelerate commit activities.

Finally, as in the original paper, we perform the t-test on the results of the values of $\tau_E$ and $\tau_R$ to see if the differences between the real and simulated series are statistically significant.

5.2 Results

Figure 5.2 shows the communication and working events for one day for one of the developers in our team. Each bar in the figure represents an event happening at that time. Visual inspection from this figure suggests that there is indeed some relation between these event sets on this day. It should be noted that this day is a typical day for this developer, it was not cherry-picked for this image.
require "rubygems"
require "csv"

['D1', 'D2', ...].each do |name|
  real = CSV.table("#{name}-incoming.csv")[:time].sort

  # Calculate deltas
  deltas = []
  real.each_with_index do |val, index|
    next if index < 1

    deltas << val - real[index-1]
  end

  # Shuffle deltas
  deltas.shuffle!

  # Add deltas to first value
  simulated = [real[0]]
  last = real[0]
  deltas.each do |d|
    last += d
    simulated << last
  end

  CSV.open("#{name}-simulated.csv", 'w') do |csv_object|
    csv_object << ['time']
    simulated.each do |t|
      csv_object << [t]
    end
  end
end

Listing 5.4: Ruby script to generate simulated in communication events
require "rubygems"
require "csv"

[ 'D1', 'D2', ...].each do |name|
  real = CSV.table("incoming/#{name}-incoming.csv")[:time].sort.map do |v|
    v - 21600
  end

  commits = CSV.table("commits/#{name}-Github.csv")[:time].sort

  t_E = []; t_R = []

  commits.each_with_index do |val, index|
    c = real.select{|v| v > val}.first
    if !c.nil? and (commits[index + 1].nil? or c < commits[index + 1])
      t_E << c - val
    end

    c = real.select{|v| v < val}.last
    if !c.nil? and (commits[index - 1].nil? or c > commits[index - 1])
      t_R << val - c
    end
  end

  CSV.open("incoming/#{name}-t_E.csv", 'w') do |csv_object|
    csv_object << ['t_E']
    t_E.each do |t|
      csv_object << [t]
    end
  end

  CSV.open("incoming/#{name}-t_R.csv", 'w') do |csv_object|
    csv_object << ['t_R']
    t_R.each do |t|
      csv_object << [t]
    end
  end
end

Listing 5.5: Ruby script to calculate $\tau_E$ and $\tau_R$
Figure 5.2: Events from the GitHub data source (commits) and the Hangouts data source (sent messages)

Figure 5.3: Events from the GitHub data source (commits) and the Hangouts data source (sent messages) for a week
Figure 5.3 shows the same data for the same developer, but now with a resolution of a week (note that the data shown in Figure 5.2 is that of the 14th). Again, visual inspection suggests a relation between these event sets.

Figures 5.5 and 5.4 show plots of the event times versus event indices for one developer for GitHub commits and sent Hangouts messages respectively. The relative straightness of these lines suggest that the processes for generating the events is stationary, a prerequisite for further analysis we want to perform.

Appendix C shows the plots for all analyzed developers individually. From these figures it can be seen that although the number of events differ, the slope is generally similar between all developers.

To formalize this indication, we perform the augmented Dickey-Fuller test \[8\] on all time series. Specifically, we used the \texttt{adf.test} function from the R package \texttt{tseries}. The null-hypothesis for this test is that the given time series is non-stationary.

For all developers, for both the Hangouts and GitHub time series, the ADF-test resulted in a \(p\)-value of < 0.01. Thus, for all time series, we reject the null-hypothesis of non-stationarity.

5.2.1 Granger causality

Twelve members of the development team supplied us with their Hangouts data dumps. Two of these dumps are from quality assurance engineers and not developers. Since we use commit events for the Granger causality test, we chose to discard these two dumps in this case, since making commits is not a core part of the job of a QA engineer.

Table 5.1 shows the results of the executed tests. For each developer that sent us their Hangouts data, we show the \(p\)-values for the causality test both ways. Formula ‘github \(\sim\) hangouts’ indicates the ability of the communication event set to forecast the work event set.

As shown in the table, we found significant results for each developer.

5.2.2 Working rhythms

We were able to get the in-communication data for eight developers. These are the same developers as in the previous experiment, with the exception of developer D5 and developer D10. When we asked developers for their Hangouts data dumps, we also sent them the script (from Listing 5.1) to generate the out-communication timestamps (required for the previous experiment) so they could generate these themselves if they were so inclined.
Figure 5.4: Event time versus event index for Hangouts message events of one developer

Figure 5.5: Event time versus event index for GitHub commit events of one developer
Table 5.1: p-values for Granger causality between work and communication activities of 10 developers in the Nubis team

Developers D5 and D10 chose to do this and only sent us their timestamps and not their full dumps. We were not able to get their in-communication timestamps before analyzing the results of this experiment.

Figure 5.6 shows the box-and-whisker diagrams that compare the results for the evaluation ($\tau_E$) and response ($\tau_R$) latencies for the real and simulated communication sets. As can be seen, both $\tau_E$ and $\tau_R$ are a lot lower for the real data set.

This figure is analogous to Figure 7 in [32].

Table 5.2 shows the values of the t-tests we performed on these results. This table is analogous to Table IV in [32].

The authors of the original paper do not mention if their values for $\tau_E$ and $\tau_R$ follow a normal distribution. However, visual inspection of Figure 7 from the original paper suggests they do not. Since our values are not distributed normally either, we perform the MannWhitneyWilcoxon test on our data [17]. The result of these tests is shown in Table 5.3. As with the results from the t-test, we can see that the differences between the real and simulated response latencies are statistically significant.

If we compare the real latencies with the simulated latencies, we see that the simulated values are much higher. This suggests that the communication activities accelerate the work activities.

5.3 Discussion

The results of the correlation between the various performance indicators and communication events allow us to answer RQ3. These results also
Table 5.2: t-Tests for the differences between the simulated and real evaluation and response latencies

<table>
<thead>
<tr>
<th></th>
<th>Simulated</th>
<th>Real</th>
<th>t-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>4.7 (h)</td>
<td>0.7 (h)</td>
<td>12.20</td>
<td>&lt; 2.2 × 10^{-16}</td>
</tr>
<tr>
<td>Response</td>
<td>5.3 (h)</td>
<td>2.1 (h)</td>
<td>6.76</td>
<td>1.726 × 10^{-11}</td>
</tr>
</tbody>
</table>

Figure 5.6: The box-and-whisker diagrams for real and simulated evaluation and response latencies
Table 5.3: MannWhitneyWilcoxon tests for the differences between the simulated and real evaluation and response latencies

<table>
<thead>
<tr>
<th>MWW-Test</th>
<th>Simulated</th>
<th>Real</th>
<th>W</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>4.7 (h)</td>
<td>0.7 (h)</td>
<td>6287400</td>
<td>&lt; $2 \times 10^{-16}$</td>
</tr>
<tr>
<td>Response</td>
<td>5.3 (h)</td>
<td>2.1 (h)</td>
<td>5707500</td>
<td>&lt; $2 \times 10^{-16}$</td>
</tr>
</tbody>
</table>

provide a jumping off point for future work: If we know that certain performance indicators are related to communication, it might be possible to improve the output of the software development process by taking away frustrations or by building better communication tools.

From the results of the Granger causality tests we ran we can see that we reject the null hypothesis in every case. This means we do note some relationship between committing on GitHub and chatting on Hangouts.

While all experiments we executed for the Granger causality test result in very low $p$-values, which cause us to reject $H_0$, which states that one time series is not a good forecaster for the other, two $p$-values stand out as being higher.

In the case of developer D8, this can be explained by the fact that this developer usually works on smaller projects, alone. This developer does not normally have to discuss with other developers about projects that they are working on, which could lead to the higher $p$-value.

Another thing to consider for developer D8 is that they sometimes work on projects that are not tracked in the Nubis GitHub organizations. Because of this, we are missing quite a lot of 'work events' for developer D8 since they are not counted as commits in our data. If we did have these events as well, it could be the case that we would also see a $p$-value of < $2 \times 10^{-16}$ for developer D8. These missing work events might explain the higher $p$-value as well.

Developer D10 does work on projects with other developers. However, this developer has a preference for delivering completely finished packages of software and usually working at night. They discuss design decisions less often than other developers. This could explain the higher $p$-value for developer D10.

Another possible explanation for the higher $p$-value for developer D10 is that the data for developer D10 is 'less stationary' than the data for other developers. Looking at Figure C.10, the lines of the index versus date for both types of events is much less straight than that for other developers. This developer has had several periods in the past where they worked less
because they wanted to give extra effort to their education. Because of this, the Granger causality test might not be applicable in this way and the results might not be usable.

Future work is needed to determine if this relation can also be shown for quality assurance engineers. We have gathered data for the communication activities of QA engineers and we have made steps towards the gathering of data for work of QA engineers. In the case of Nubis, this means specific changes to the states of issues in the tracker. Accepting or rejecting issues is a core part of the work for QA engineers. Issue state changes can be found using the GitHub events API.

Comparing our results for the second experiment with those from the original paper, we see that we come to the same conclusion: The average real evaluation and response latencies are 0.7 hours and 2.1 hours, much shorter than the simulated values of 4.7 hours and 5.3 hours. For both these values, we found that the differences are significant.

One thing we note is that the values for the evaluation and response latencies found in [32] are much larger than the ones we found. We theorize that this might be due to the differences in medium. The communication data we used to run the experiment is from Hangouts, an instant messenger while the original authors used email archives for the communication data. In general, instant messaging platforms have a higher volume of messages than email does. This might make this method less applicable for analysis of communication in the form of instant messaging.

Another thing we note is that the method proposed in the paper makes some general assumptions on what messages could mean in relation to commits. Since we have much more well-structured data available than the authors of the original papers (for example due to the traceability requirement introduced in Section 6.2.2) we might be able to actually determine if there is a relation between a certain commit message and a certain in-communication event. For instance, communication events from group projects chats not related to the repository the commit was made to can be discarded for this analysis. Future work is required in this area.

We know from Chapter 4 that committing is a vital indicator of performance for a virtual team. From our results in this chapter, we conclude that communication and work activities in a virtual team are related.
5.3.1 Evaluation

We presented our results from this chapter as part of a set of internal talks at Nubis. During discussion, attendees raised a couple of possible threats to validity and questions.

*Do you think gifs sent in group Hangouts impact your results, as you assume most messages sent are work related?* No. In most project related chats, the ratio of 'fun' messages versus work related chats or notifications is very much in favor of the work related content. For the few developers that are a part of group chats with colleagues that are not work related, volume in those chats is much lower than that in work chats.

*In the Working Rythms experiment, what would happen if a developer sent a message to a group chat just before committing?* If the developer sends a message and immediately gets a response before committing, that last response will be counted for the response latency. One can argue if, in that case, the commit is actually a response to that message, but that is how the response latency is defined.

*Have you looked at a reason for the relationship between communication and work?* No. Our data only suggests a correlation, we can not show causality or a possible reason behind that.

5.4 Threats to validity

We identify several threats to construct validity of the experiments performed in this chapter. First, as is clear from the graphs in Appendix [C](#) not all developers have the same work patterns over the time we analyzed. Since the method we used to perform the analysis depends on the data being from a continuous process (i.e. developers working at more or less the same rate throughout the entire time period) this means we might get worse results for the developers that have had periods in which they work less in the time we analyzed. This might especially be the case for developers D6, D8, D9 and D10. However, the results we found for these developers are similar to the results for the developers that did work continuously.

Another threat to construct validity can be found in the second experiment. As stated in the discussion, this experiment was designed for email communication and might not be applicable to our situation, in which we use instant messaging.

We have identified one internal threat to validity in the first experiment. The size of the bin we choose for turning the event stream into time-series
data has a huge effect on what the resulting time-series looks like. We chose one minute since it is small enough to still show order between GitHub and Hangouts events in the data, but large enough to be able to analyze. However, this method reduces the amount of information we have available so applying it could cause patterns to emerge in the data that were not there before or remove existing patterns in the data altogether.

As with the previous chapters, we identify an external thread to validity in that the Nubis team must be a typical virtual team if we want to generalize our results to all virtual teams.

In this chapter, we have seen that there is a relation between communication and work activities in virtual teams, as there is in open teams. Moreover, communication activities accelerate work activities. In the next chapter, Chapter 6, we use this relation as motivation for improving communication tools.
Chapter 6

Improving tools and measuring the results

In this chapter we look at RQ4: How can we improve communication tools for virtual teams and how can we measure their impact?

6.1 Methodology

To answer this question, we will conduct follow-up discussions on the results from the interviews conducted in Chapter 4. In these discussions, we will start with the concerns raised in the interviews and try to formulate them into a list of problems we want to address with improvements to the communication tools used at Nubis.

6.2 Results

6.2.1 Identified problems

We have identified a list of problems we want to address. For each problem we list the source and motivation behind it. If a problem originated in the interviews from Chapter 4, we state the concern number (as labeled in the interview results) that it is formulated from.

P1 It is hard to compare work committed with hours written in timesheets with the current systems
    Tim and Duco, from interviews (concern I7, part of concern I5)

P2 Notifications from GitHub are not enough to keep track of project board changes
    Philip, from interview (issues as productivity indicator, concerns I1, I6) and follow-up talks
P3 Performing analysis on hours worked is hard within the hour tracking system, any non-trivial question can only be answered by first exporting the data to some other program
Duco, interview (concern I3) and follow-up talks

P4 Keeping track of running projects requires either tools which do not integrate with the rest of our tools or abuse of my mailbox
Philip, follow-up talks

P5 As the team grows, it gets harder to keep track of everyone and their current positions
Tim, introspection

P6 There is no structured place to keep track of the history of a developer at the company
Tim, introspection

P7 Planning in students working from home for community management shifts can be risky
Jorrit Leemans (manager of the community management team), talks about automation within the company

P8 There is no way to enforce commit traceability requirements
Tim, introspection, literature (being able to relate items in different software artifacts is a requirement for many software engineering tasks [1])

P9 Deployments are not visible to everyone working on a project
Philip, follow-up talks

P10 It is easy to forget running and upcoming projects when they are not causing issues
Tim, introspection

P11 Working with students, it can be hard to get a good overview of a developer’s availability
Philip, interview (concern I2)

P12 Full-time employees are not completing their timesheets, making it hard to get clear numbers on project costs
Duco, email received with problem stated, hinted in interview (concern I4)

P13 There is no central place where developers can find all info related to a project
Tim, introspection, literature (research suggests the negative effect of geographic distance on coordination is mitigated by sharing knowledge [9])
It is not possible for recruits to register possible recruits in our existing systems.

Philip, follow-up talks

Concern I5 from the interviews (being able to identify which people are working too much) is only partly represented. Initially, Duco was also interested in the moments of the day employees are using Hangouts. After follow-up discussions with Duco, we ended up only considering the hours that employees actually log, out of privacy concerns.

6.2.2 Solutions

In this section, we describe the projects we have developed in preparation of and during this project and the problems they address.

Nacho

As described in Section 2.4, Nubis uses Nacho as a general chat bot for a lot of different activities. Originally, we experimented with using Slack at Nubis instead of Hangouts. However, due to the way most people at Nubis use Hangouts (mobile usage and heavy focus on multiple chat windows open at one time) and the fact that the development team works together with people from the other (less technologically inclined) teams (for which Slack turned out to be less well suited) we decided that Slack was not a good fit.

Instead, we focused on bringing the feature from Slack we wanted most - the ability to add our own bots and integrations - to Hangouts instead. The result of this is Nacho.

Nacho works by using the (unofficial, undocumented) chat API intended for the Hangouts chat window in Gmail. It provides a wrapper around this very low-level API and offers a plugin interface.

Since its inception, a plethora of plugins has been developed - some useful, others just for fun. We want to highlight a few.

github

As described in Section 2.4, Nubis uses conventions on issue labels to create a project board. This project board can be visualized using third-party tools, such as Waffle\(^1\).

Figure 6.1 shows a typical project board. All projects at Nubis that use this approach to project management (some of the smaller, one-off projects do not) use the same conventions for columns (and thus, labels). Moving an issue from the ‘Backlog’ column to the ‘Ready’ column means developers should start working on it. Once they do, they move the issue to ‘In Development’. If a developer finishes the feature or fixes the bug they move the

\(^1\)https://waffle.io/
issue to the ‘Test’ column. If they do not and stop working on it, they move it back to ‘Ready’, for another developer. A quality assurance engineer can then test the functionality related to the issue and reject it by moving it back to ‘Ready’ or accept it by moving it to ‘Accepted’. From there, it is eventually cleaned up by being moved to ‘Done’ and being closed.

While this structure is great for getting a snapshot of the state of a project, it does not work well with real-time updates as systems as GitHub are not setup for notifications based on label changes. To combat this, we developed a Nacho plugin that is set up to receive GitHub webhooks when changes are made to issues and that sends messages to a certain group chat (based on the repository the issue that was changed is associated with) with what happened. The plugin knows the meaning behind all label changes (as described above) so it can generate rich notifications.

In Figure 6.2 an example project chat with the github Nacho plugin enabled is depicted. This project chat also has the review plugin enabled, which asks for a code review in the project chat each time a pull request of a feature branch to the master branch is opened.

The github plugin solves Problem P2.

untraceable For projects that use the structure above, a requirement is placed on the developers to make sure each commit made can be traced to an issue. This enables us, looking back, to always identify the reason behind a single line change, be it a feature request or a bug report. This is implemented by including a line ‘For #issuenumber’ in the body of a git commit message. Using this syntax means that the commits will show up within GitHub’s issue detail view as well.

To enforce this measure (which is easily forgotten) we developed a Nacho
Figure 6.2: Hangouts chat showing the Nacho github plugin

The plugin listens to a webhook from GitHub that fires whenever a commit is pushed. If the commit message body does not include an issue, Nacho sends a message to the project chat group, publicly shaming the transgressor. This message is visible in the group chat in Figure 6.2.

This plugin solves Problem P8.

Another plugin uses this traceability requirement to send a message to a group chat with all quality assurance engineers whenever a pull request is opened that contains a commit that has not yet been accepted.

**ship** The ship plugin is set up to receive a webhook from Codeship (which Nubis uses for continuous deployment as discussed in Section 2.4) whenever a deployment is started, failed or completed. Usually, this plugin ignores these messages. However, if a developer breaks the master of a project (for instance by introducing code which does not pass the linter or does not pass tests), Nacho sends a message to the group chat of that project so other developers can react to the situation.

If a deployment to the production server is started, Nacho will send a message to the group chat indicating this, so everyone that interacts with the project is aware of this. Once the deployment finishes (or fails!), Nacho will notify the project chat of this. Usually, someone in the chat will then open the project on the production server to make sure nothing went wrong.

The ship plugin solves Problem P9.
The communitymanagement plugin was the first plugin to be created for another team. The Nubis community management team responds to messages sent to the Facebook pages of large brands outside of office hours. For managing scheduling of this, they use a Google Spreadsheet with, for each day of the month, the name of the person that is scheduled to work. Since the community management team also works with students, there have been a few incidents where people forgot their shift.

To avoid this in the future, the Nacho communitymanagement plugin reads the spreadsheet and checks who is scheduled to work that evening. During the afternoon, this person will receive a heads up reminder from Nacho that they are expected to work that evening. Ten minutes before the start of their shift, Nacho sends another message asking for a confirmation that they are indeed ready to work. If this confirmation is not received, Nacho will notify the community management group chat that no one is covering the current shift. At that point, someone else can step in or try to contact the person that was supposed to work that evening.

This plugin solves Problem P7.

After we obtained the results of the survey (discussed in Chapter 3), we released documentation on how to write Nacho plugins. Using this documentation, Nubis developers can build their own plugins and have them added to the company Nacho installation. While the source code for Nacho is proprietary, we have included the full documentation in Appendix D as it gives a good overview of the power of Nacho plugins.

verm

To have a centralized place for the data we collect and to make it easy to get insights into this data, we developed verm. The initial goal behind verm was to give an overview of everyone in the development team, but that was expanded once more data sources became available to us. For instance, we added a record of all commits pushed to GitHub within the organization to verm (as mentioned in Section 5.1.1).

verm is available to all departments within Nubis, but was initially created for the development team, and is thus best equipped to deal with a team of developers. Note that access to verm is restricted to department heads.

Currently, verm is mainly used to see how a team is performing. The verm dashboard view is shown in Figure 6.3. This screen gives an overview of the hours worked in the last 30 days, organized by both projects and developers. The dashboard provides a preliminary solution to Problem P3. We are working on adding more advanced reporting options.
Figure 6.3: Dashboard view of *verm*, with as main plots the hours worked by project and developer in the last 30 days

Remember that most developers at Nubis have a 0-hours contract. This means the hours they work can vary greatly from day to day (or even from week to week). The dashboard view shows if the hours worked in the last month are stable, or if there are clear peaks (which there are in the snapshot in Figure 6.3).

Next to the main graph, the total number of hours worked last month (and the associated labor costs) are shown. This tool can be used by Duco to answer the questions he has about team productivity. It also helps in discovering if employees have a healthy work-life balance by looking at the number of hours logged. While not perfect, this metric should give an initial indication that can be used as a starting point for talking to people.

We are working with Duco to further develop the reporting functionalities in verm to be even more useful. Since verm can combine data from multiple data sources (such as the hour tracking system and commits pushed to GitHub) in real-time, we are confident that verm can generate reports that are much more valuable than traditional spreadsheets.

The dashboard also includes information about the number of currently active projects and the ones that might be upcoming. These serve as a reminder to follow-up on any outstanding actions.

Furthermore, the dashboard shows which developers are not currently assigned to a project which is active.

In Figure 6.4 verm’s developer detail view is shown. This screen can be used to get insights about a single developer’s working habits in the team.

It shows the developer’s current level, their salary, business indication
and projects they are assigned to. The business indication is a metric which shows if a developer is: available, busy or swamped. Currently, this metric is defined by simply looking at the number of active projects the developer is assigned to, but we are working on improving this by looking at the average hours this developer works.

Furthermore, verm users can leave comments on team members, and view comments by other users. This is used to keep track of an employee’s growth within the organization.

Next, a graph of the hours this developer worked in the last 30 days is shown, segmented per project. This is a similar graph to the one on the dashboard, but filtered by just the one developer. Above this graph, the total number of hours and associated labor costs are shown.

This page also shows a trend line of the hours this developer worked in the last 10 weeks. Since employee planning can be a challenge due to the fact that most developers at Nubis are students, the information on this page (combined with the busyness indication) can be a first aid in figuring out who can work on an upcoming project.

The developer detail page solves Problems P5, P6, P1 and P11.

As indicated on the dashboard, verm also stores information about all projects that the team is working on. On the project detail page, users can associate employees in different roles to the project. They can also link the project board and all repositories that belong to the project. An about text (more on this in the next section about hub) is shown, together with a list of recent commits and an overview of the hours spent on this project.

This can be used to get a quick update on how a project is progressing.
with regards to planning and spent budget.

Another functionality of the project overview in verm is the review page. Team managers use this feature to hold a weekly review of all active and pending projects. The system loops through all projects (in order of latest commit activity), at which points the managers make sure they know all outstanding actions and are aware of any upcoming deadlines. After showing active projects, the system moves on to pending projects, for which a follow up action might be required in the coming week. The review feature solves Problems [P2] and [P10]

Another feature of verm which was requested by Nubis management is the integration of the hours tracking system to check if all employees are completing it. Each hour, verm downloads a CSV formatted export from this system and parses it into its own data structures. Then, for all full-time employees (which should be logging hours each working day), the system checks if the employee has entered any data for the last two working days. If they have not, verm sends a message to a webhook plugin running in Nacho. In turn, Nacho notifies the employee that they have not filled in any hours in the last two days.

While annoying, this measure has helped improve the completion of time sheets for Nubis.

This feature helps solve Problem [P12]

This feature was later extended to include the reporting of hours worked described above.

**hub**

Nubis uses a lot of different services for different aspects of project management. To give people working at Nubis a centralized overview of all resources for a project, we developed **hub**. hub is a front end for the data in verm, scoped to the projects the logged-in user can see.

Figure 6.5 shows a project detail page in **hub**. All data shown on this page is loaded from verm. Users get to this page from an overview page (not pictured here) which lists all the projects they are assigned to.

People that are associated with the project with the project manager (PM) role can edit the about text, which usually contains general information about the project. This can include server addresses, where to find login credentials, testing procedures or even a high level project planning.

Besides this free-form about section, projects on hub have links to their project board (which links to a page as shown in Figure 6.1), repositories on GitHub and project URLs. A field for code search is available which links to an external service which indexes all of Nubis’s GitHub repositories.
Performing code search from hub immediately links to a search result within the repositories that are associated with the project the employee is viewing. Finally, hub lists all people working on the project.

hub solves Problem P13

novi

novi is another front end to verm. It is intended to be used by recruiters, people in the team that try to bring on other developers.

Figure 6.6 shows the screen recruiters use to register new possible recruits in verm, using novi. When they do this, the recruit is added to the ‘Recruits’ screen in the verm front end as well. Additionally, history items (as shown in Figure 6.4) are created for the date that the recruit was registered in the system and the date for the first interview.

Using the recruit detail screen, shown in Figure 6.7, recruiters can add comments to recruits and log information about planned interviews. Note that this screen corresponds with the developer detail screen from Figure 6.4. Once a recruit is hired, their profile moves from the recruits page in verm to the developers page, and they disappear from novi.

novi solves Problem P14

We are currently working on a new ‘Work at Nubis’ page for the Nubis website which includes a form where applicants can enter their information to get an interview. When someone submits this form, that data is passed directly to verm and the person is added to the system as a recruit. Then,
Figure 6.6: Registering a new recruit in *novi*

Figure 6.7: Viewing a recruit in *novi*
verm sends a message to a Nacho webhook plugin that triggers a message to all recruiters that have access to novi with a link to the new recruit in novi. A recruiter can then contact the recruit to set up an interview.

**TMH-ext**

To encourage employees to complete their time sheets, we developed a browser extension for Chrome that improves the user experience on the service that Nubis uses for hours tracking. A lot of employees at Nubis have installed the extension (as of August 2016 the extension is used weekly by 22 employees) with positive feedback.

The extension helps solve Problem P12 by making it easier for people to complete their timesheets.

Due to the success of the extension and the fact that verm is now equipped to handle hour tracking data (due to the automated data imports) we are considering creating our own simple time tracker that is integrated into verm.

### 6.3 Discussion

In this chapter, we have identified a number of problems with communication and the virtual team at Nubis. For each of these problems, we have offered solutions.

Below, for each of the identified problems, we list the project or projects that address that problem.

- **P1** verm
- **P2** github Nacho plugin
- **P3** verm
- **P4** verm
- **P5** verm
- **P6** verm
- **P7** communitymanagement Nacho plugin
- **P8** untraceable Nacho plugin
- **P9** ship Nacho plugin
- **P10** verm
- **P11** verm
By creating these tools, we have made it easier for Nubis to start measuring their productivity (on all three levels of productivity as found in Chapter 4). We have also taken preliminary steps towards implementing better communication tools at Nubis. That work will have to be continued after this thesis, at which point the dashboarding tools we have implemented can be used to measure the impact of these changes on productivity.

### 6.4 Threats to validity

Since the Nubis team is relatively small and has few management-level positions, we only talked to a few people to identify the problems listed in this chapter. This could be a threat to construct validity, as this low number of interviewees might not be able to give a comprehensive list of problems. However, the interviewees were aware of the survey results (from Chapter 3) during the follow-up discussions.

The problems defined in this chapter are very specific to the situation at Nubis. Therefore, we do not think the tools we have developed to solve them can be generalized easily to other teams.

We do, however, think that the process ideas behind these tools can be applied to other virtual teams as well.

In this chapter we looked at a list of problems relating to communication tools or measuring productivity in the team. We have developed several tools that address the problems in this list.
Chapter 7

Discussion

In Chapter 3, we concluded that, for core team collaboration and work activities, virtual teams use communication tools for the same purposes as open teams. We saw that open teams have a broader range of use cases for communication tools than virtual teams do.

From Chapter 4 we learned that while virtual teams have more sophisticated indicators of productivity available to them, counting the number of commits made is a valid indicator of productivity for both virtual and open teams.

Using the results from these chapters, we compared work and communication patterns for virtual and open teams in Chapter 5. We were able to find a relation for these activities in our team, and found results similar to an existing study on open teams.

Finally, in Chapter 6 we identified a number of problems and bottlenecks in the communication tools used and Nubis and the ability to measure productivity. We proposed solutions in the form of tools to these problems.

Combining this knowledge, we conclude that improving communication tools will have not only a positive effect on the satisfaction of a virtual development team, but is in fact a requirement for reaching the maximum possible productivity of a team. Furthermore, we conclude that it is possible to use results obtained for open teams to improve communication tools for virtual teams.

However, we note that it is possible to go a step further for virtual teams. Our work provides a jumping off point for further research into leveraging the extra information available for virtual teams to develop better tools. During our research, we have set up preliminary data collection for this purpose. Nubis would like to welcome a Master’s student to continue this research.
Bibliography


[27] Margaret-Anne Storey, Leif Singer, Fernando Figueira Filho, Alexey Zagalsky, and Daniel M. German. How social and communication channels shape and challenge a participatory culture in software development. 2016.


Appendix A

Survey

A.1 Introduction

Hi, As some of you may already know, I’m doing research to figure out how communication works in virtual teams (like ours).

In this survey, I want to ask you a few questions about Hangouts. Your answers will be only be used in this research project and to improve communication tools.

It should only take about 10 to 15 minutes.

Results from the survey will only be published in anonymized (“developer 4 said ...”) or aggregated (“developers that have been at Nubis for at least 2 years ...”) form. Your response will be handled confidentially. Once I have analyzed the results I will share the outcomes with all of you.

If you have any questions about the survey itself or how I’m handling the data, please don’t hesitate to contact me at tim.vandalen@nubisonline.nl.

Tim

A.2 Questions

A.2.1 About you

a. How long have you been using Hangouts? (It’s been around 3 years since GTalk became Hangouts)

b. How long have you been at Nubis?

c. What is your primary role in the software development process?

- Developer
- Project Manager
- Designer
- Other
A.2.2 About Hangouts

a. How often do you use Hangouts for working?
   - Frequently (most of the workdays)
   - Sometimes (some of the workdays)
   - Seldom (only a few times)
   - Never

b. How many Hangouts group chats related to a software project are you in?

c. What do you use Hangouts for?

d. Which features of Hangouts do you use for work?
   - Project group chats
   - Direct chats
   - Non-project group chats
   - Nacho plugins
   - Image upload
   - Gmail chat log search
   - Emoji

e. Which features of Hangouts are most important to you for work?
   - Project group chats
   - Direct chats
   - Non-project group chats
   - Nacho plugins
   - Image upload
   - Gmail chat log search
   - Other

f. What benefits do you get by using Hangouts?

g. What do you get frustrated with while using Hangouts?

h. What features to you think Google needs to add/improve? Why?

i. Do you use any other instant messaging channels (for instance Slack, HipChat, Facebook Messenger, WhatsApp) for software development? Why (not)?
A.2.3 About Nacho

Nacho plugins can do two things: they can allow external services (like GitHub or Codeship) send notifications and they can create commands to run automated tasks.

a. Which Nacho plugins do you use on a regular basis
   - !afwas
   - !community
   - !echo
   - !emoji
   - !gif
   - github (issue updates)
   - !image
   - !ping
   - !punteen
   - !random
   - reaction (o.a. !wow, !pika)
   - ship (codeship deployment updates)
   - trash (vuilnis dagen)
   - untraceable (commit mistake notification)
   - tmh (behind on hours notification)
   - !weather

b. Please indicate the plugins you use most frequently and why you use them

c. Can you think of any improvements you would like to see made to the existing plugins?

d. If anything is possible, what kind of plugins would you like to have beyond the existing ones?

A.2.4 Would you like to comment or add anything else?

That’s all! Thanks for your time, if you have any comments, be sure to let me know.
   I will distribute the outcome of the survey once it’s ready.
   Tim
A.3 Emails

A.3.1 First email

Hi all,

As you may know, I’m currently doing research in the space of virtual software development teams, like ours, for my Master’s thesis. As part of this research, I want to know how everyone at Nubis uses Hangouts.

Please take some time to fill in the survey: 
https://nubisprocess.typeform.com/to/vZfWSp

The survey is aimed at getting a better understanding of how we use Hangouts to facilitate our work and, ultimately, improving our communication tools. If you’re interested in my research, here is a short presentation: 
http://timvdalen.nl/files/kickoff/presentation.pdf (I can imagine some people would like me to get good results for RQ6!) Or, feel free to email/talk to me about it anytime.

Met vriendelijke groet,

Tim van Dalen

A.3.2 Reminder

Hey all,

Just a reminder to those of you that have not filled in the survey yet. Please do so before 24th, as I want to start analyzing the results then.

https://nubisprocess.typeform.com/to/vZfWSp

If you’ve already completed it, you can ignore this email.

Thanks!

Met vriendelijke groet,
A.3.3 Developer reminder

Hi devs,

I’m still missing some of your responses to my survey. Please take some time to complete it, I really need your input!

https://nubisprocess.typeform.com/to/vZfWSp

Met vriendelijke groet,

Tim van Dalen

A.4 Response dates

Most respondents completed the survey immediately after receiving an email about it.

A.4.1 Up to first reminder for all

Figure A.1: Submission dates for the survey results submitted after the first email. Role 'None' means that the respondent does not have a position that is directly related to the software development process.

Figure A.1 shows the response dates for all respondents that completed the survey before the first reminder email. The initial email seems to have reached mostly people who felt they play an active part of the development process.
A.4.2 Up to reminder for developers

Figure A.2: Submission dates for the survey results submitted after the first reminder for everyone

In contrast to the first email, the reminder email reached the people who did not feel like they play an active part in the software development process.

A.4.3 After developer reminder

Since our focus will be on developers for most of the next two chapters, we wanted to make sure that we had a good overview of how the development team uses communication tools. Because of this, we decided to send another reminder email to just the development team.

Figure A.3: Submission dates for the survey results submitted after the reminder for developers

Figure A.3 shows the response dates for the results submitted after this email. One developer filled in the survey immediately after receiving the email while one quality assurance engineer filled it in about two days later.

After this, we exported the results and stopped accepting new submissions.
Appendix B

Interviews

B.1 Duco ter Steege

Tim: Okay, good. So for you, looking at productivity of the team is more of an overview thing. You look back at the end of the month, I guess, or just weekly. You have some way of getting to know for yourself how productive the team has been. So my question to you is what do you use as indicators for this productivity, the fact that work has been done?

Duco: I actually just look at the numbers, at the Euros. While we’re still in the process, but I’m trying to find different parts, so we have a developments team, we have a design team, we have a search team, and I try to make a balance of what comes in and what goes out as salary. And there should be ... there should be more Euros coming in in one end and a healthy amount should go out on the other end. So that’s actually what I ... if the balance is okay then I think that the bigger picture the business unit is doing well.

Tim: Okay. So that’s looking back. Do you try to keep up to date in a more real time passion or do you just usually look at the bigger picture?

Duco: Well, we’re at a point with Nubis right now that it’s almost impossible for me to check everyone if they are productive on like a weekly basis or something. So we’re trying to put that at ... well, for instance for development. You and Philip are responsible to maintain a health balance to it. There are a lot of more things that you have to worry about, so I try to talk to people who are in somewhat management layer to see how things are going.

Tim: So real time it’s more talking to people that way rather than looking at ...

Duco: Yeah, I do check hours regularly, but not ... but more if people fill in their hours. And again I don’t try to check everyone’s hours because it would be a lot of people that I have to check a lot of time. I try to check the people who I think should be checking other people because I think they
have to set an example. So yeah, that’s one of the things I try to.

Tim: Okay, so I guess it’s safe to say there for you the main indicator is
the salary for the students in our development, that just means the hours
worked. That’s your main indicator for productivity for the development
team?

Duco: Yeah.

Tim: Okay, good.

Duco: So salary and then compare it with everything that we built. So
like salary for the past three months and all invoices from the past three
months or from the last month. But as always you send an invoice after
work is done and not before work is done. So there’s always a difference in
there, but that’s somewhat what I’m trying to match.

Tim: Okay. Good. So that’s for productivity. Are there any other
things you look at or would like to look at as some sort of indicator for how
the team is doing. So just in general any other things besides productivity
that you think are important and that you look or that you want to look at.

Duco: I think there’s always a lot of things to look at, but I can’t really
come up with something really specific. But that’s just a question of talking
to people and part of my job is to ask to certain people if how it’s going
with new projects and if new projects ... the new people they talk to if
there’s going to be a project, new project from that if we’re all at progress
in projects that’s going. And so that’s more talk to people and from that I
get information if things are going well or if they need help and kind of ... it
depends who we’re talking about. But I know from the amount of questions
I get from certain people if things are going well or not. If I don’t get any
questions, that could be a sign that it’s not going well. But that could also
be a sign that things are going well. But I have to ask and in some cases I
know that if I’m not hearing new things that I should be asking questions.

Tim: Yeah, okay, so earlier we discussed work-life balance. So you men-
tioned being interested in patterns of being online, being working basically.
Could you talk a little bit about that?

Duco: Well, we have a culture that you can work at any moment at any
place, certainly for developers. I think it’s interesting to know how people
experience that. If they experience that opportunity as they have to work
all the time or they can work all the time and there’s a difference. I think
for ... especially for people with flexible contracts, flexible hour contract,
it’s really nice that they can work whenever they want. But it could be
that they experience pressure if for instance you or Philip ask them to do
something, if they would be ... should be doing other things. For people who
have a full time contract that might be the same cause maybe some people
don’t really like being asked questions at let’s say at 9:00 in the evening.
That’s interesting to know and maybe from your data we can get something,
going to get the information about that.

Tim: Okay, so just looking at when certain people are online and talking
if that’s ... well, I wouldn’t want to say too much. But the amount of hours someone is general active during the day would be an indicator for you to maybe talk to them.

Duco: Yeah, if someone with a full time contract is talking online almost every evening in the week I would want to know what they’re talking about, do they need to feel the need to keep working or they just like to check with other colleagues more informally or it could be a lot of things.

Tim: Okay. Good. So that was very specific example of an indicator that you can use to make decisions basically. So any others come to mind or that we ...

Duco: No, not right now.

Tim: Okay, then I think that’s what I wanted to know. Alright, thank you.

Duco: Great.

B.2 Philip Vermeij

Tim: Part of the team works evenings and nights. When you get to the office in the morning what do you use to see how productive the team has been since you’ve last checked in? Just looking back at the previous days and nights, what do you use to see how much work has been done?

Philip: At first the get up emails in my inbox and simultaneously with the natural check, but I must admit that some of them I’ve muted at this point. So above the hangout chats it’s the emails and checking at the same time with Github itself.

Tim: So what do you look at in the emails?

Philip: About which issues, what it is about.

Tim: So just the issues that’s been worked on?

Philip: Yeah, cause those were the issues that’s are sending out emails. And some of them are multiple emails at the same time, but mostly just checking the subjects and let’s say 50

Tim: But it’s always the email that lead you to Github, you don’t go looking around Github to see?

Philip: Yeah, I do that. But not the first thing in the morning.

Tim: Okay, so that brings me to my next question. Just during the day what do you use to keep up to date with how the project is going?

Philip: Well, I think it’s more of the opposite. Although the emails, I’m getting the notifications at the moment something has been done, so that’s quite useful. But I think that during the day I mostly use Github itself and Nacho cause in the morning I don’t have all the different project chests open. But as the day progresses it’s more and more opening up on my screen and I can easily see if someone is working on something, someone
Tim: Okay.

Philip: So that shifts from mail to Nacho actually.

Tim: Okay. But it’s safe to say that all your productivity measures kind of revolve around issues?

Philip: Yeah.

Tim: Okay.

Philip: Yeah, and okay, so maybe I see what you’re getting in. But issue specific it’s around issues, so Github and Nacho. But further discussions are also happening through hangouts. So during the day that’s depending on the amount of issues.

Tim: Okay, but you never look at just all the commits that were made in the last day or something like that? It’s always from the issues.

Philip: Yeah, because when something is done.

Tim: That’s what’s interesting. Okay, good. So I think that covers seeing what the productivity is. Are there any other indicators you use for the team that are important to you. So besides just productivity, are there other things you look at like team health, whatever, that are important to you?

Philip: How do you mean specifically?

Tim: So just maybe ... so checking if the team is productive is one thing. So you do that by checking the issues. But do you also look at maybe who’s working or if people aren’t working too much?

Philip: Yeah, everything can be seen from the issues, of course, from Github.

Tim: Yeah, but just to you personally is that important or do you not really ...

Philip: Well, in some way it is because based on how much someone is working you can actually think, “Okay, this has to be fixed within the next hour. This guy is always online, I’ll ask him because I know he’s always online based on previous work I’ve seen from him.”

Tim: So you look at someone’s history and you use that as an indicator for if they’re going to be able to fix things fast when you needed basically?

Philip: Yeah, and that’s of course we learn I should go with this.

Tim: And that’s ... okay, sure, any other things that you maybe look at like data indicators about the team that you use in some way to make decisions?

Philip: Well, I think it’s hard to say because ... so most of this ... some of these things are ... how do you say this? In your head. You just know these kinds of things.

Tim: Yeah, sure.

Philip: And I think when you mention them against these, the facts we have from all the boards and issue with things, I think those things you
have in your head are actually quite the same as the number say. But I
don’t actually use some kind of system that indicates this, although in the
last weeks we actually had just started paying closer attention to the hours
someone says they’re working and hours they should be writing. So maybe
I think in the future, in the couple of coming weeks and further that will
become more important. But again combined with what you know, what
you have in your head, and with the other issues say.

Tim: Okay, good. That’s it then. Anything else you want to add?
Philip: No. Nothing at this point.
Tim: Okay, cool.
Appendix C

Work and chat event graphs

This appendix shows the event index versus event date graphs that are used in Section 5.2 for all developers.
Figure C.1: Graphs for developer D1

(a) Event time versus event index for Hangouts message events

(b) Event time versus event index for Github commit events

Figure C.2: Graphs for developer D2

(a) Event time versus event index for Hangouts message events

(b) Event time versus event index for Github commit events

Figure C.3: Graphs for developer D3

(a) Event time versus event index for Hangouts message events

(b) Event time versus event index for Github commit events
Figure C.4: Graphs for developer D4

Figure C.5: Graphs for developer D5

Figure C.6: Graphs for developer D6
Figure C.7: Graphs for developer D7

(a) Event time versus event index for Hangouts message events
(b) Event time versus event index for Github commit events

Figure C.8: Graphs for developer D8

(a) Event time versus event index for Hangouts message events
(b) Event time versus event index for Github commit events

Figure C.9: Graphs for developer D9

(a) Event time versus event index for Hangouts message events
(b) Event time versus event index for Github commit events
(a) Event time versus event index for Hangouts message events
(b) Event time versus event index for Github commit events

Figure C.10: Graphs for developer D10
Appendix D

Writing your own Nacho plugin

In this document I’ll describe what’s needed to write your own Nacho plugin, and get it up and running.

First, some terminology. When I say Nacho in this document, I mean the Nacho software, not the actual running bot instance that we all know and love. So, what is Nacho? Nacho is a wrapper around Google Hangouts that makes it easy to implement bot-like behaviour. It is extensible using plugins and provides a high level API for communicating with users or external services.

D.1 Getting Nacho up and running

Nacho, and its plugins, is written in JavaScript. To be able to run Nacho, make sure you have node and npm installed on your system.

You can download the latest version of Nacho here: removed
Unpack it somewhere and run npm install to get all dependencies.

Next, you have to figure out how you’re going to run Nacho. Nacho needs a Google account to run and it won’t respond to itself, so you need to run it on a separate Google account to test it. You could for instance use an @gmail account to run Nacho on, and then test it from your @nubisonline.

Before you start, you should tell Nacho that you’re the person running it. Set the NACHO_ADMIN environment variable to your email address. You can do this one time by running export NACHO_ADMIN="you@nubisonline.nl", or by adding that line to your .bashrc or .bash_profile files (or whatever profile file you load from).

To start Nacho, run node main.js from the nacho/ folder. Follow the instructions in your terminal to authenticate the account you want to use with Nacho. After everything is set up, you should get a Hangouts message saying Nacho has started.
D.2 The plugin architecture

Nacho plugins can provide extra functionality using two mechanisms:

- Commands (!dosomething argument)
- Hooks (External services making a HTTP request)

Plugins are distributed using plugin modules. A Nacho plugin module is a CommonJS module which returns a function that takes Nacho’s plugin types and returns a Plugin class.

This might sound a little daunting, but I’ve included an example echo plugin with the Nacho distribution. Let’s start by examining the file structure:

```
plugins/  # This folder contains plugin packages
  echo/   # This is a plugin package
    package.json  # npm package definition
    index.js     # Main plugin file
```

As you can see, the package can be a regular npm package, allowing you to define all sorts of dependencies, and to use any library you want within your plugin.

Go ahead and open index.js. As you can see, the package exports a function that gets the plugin types variable, and returns a class that subtypes Plugin.

If your Nacho instance is still running, you should be able to send !echo testing to it to see if the plugin works.

Nacho automatically loads any plugin packages it finds in the plugins/ folder. Don’t forget to re-start Nacho if you change or add a plugin!

Instead of just one plugin, a plugin package can also define multiple plugins. In that case, the package should export an array of functions that each return a Plugin class.

D.3 Writing a plugin

First, decide which type of plugin you’re going to write. If you want to just respond to commands, you’ll want to extend the CommandPlugin class. If you want to listen to external hooks, you need the HookPlugin class. To mix both, use the InteractiveHookPlugin class.

All plugin types require you to give your plugin a unique name in the super constructor.
D.3.1 CommandPlugin

A commandplugin has three relevant properties:

- **commands**: `Array[String]` of commands this plugin responds to. For instance `['command1', 'command2']`

- **onCommand**: `function(message)` This high level function is called when a command is received but you haven’t defined `onMessage`. You’ll only receive the arguments to the command, not the command itself.

- **onMessage**: `function(message)` Slightly lower level, this received the message including the command called. If you plugin defines multiple commands, you must use this one to be able to distinguish them.

See the API reference for an overview of the Message object.

D.3.2 HookPlugin

A hookplugin has only one property you need to define:

- **middleware**: `function(req, res, next)` This function will be called when Nacho receives a POST request on `/hooks/:name` (the one you set using the super constructor). `res`, `req` and `next` are Express objects.

D.3.3 InteractiveHookPlugin

A combination of the command and hook variations, which takes both their properties.

D.4 API reference

To actually send messages to people and group chats, Nacho gives you a couple of different APIs to work with.

If you have a CommandPlugin, you’ll get a Message object with every incoming message. All plugins get a number of utilities they can use to send messages directly or obtain a reference to a Conversation.

Any message that sends or receives text can do so in either plain text, or as Hangouts segments. This is a special data format that allows you to add bold, italic, etc to your messages. The hangupsjs npm package includes a MessageBuilder class that allows you to build segments without having to do it by hand. Visit its documentation here[^1]

[^1]: https://github.com/algesten/hangupsjs#messagebuilder
D.4.1 Message

Represents a message sent in a chat

```
contents
```

Contents of the message, \{segments, text\}. text is plaintext, segments are Hangouts segments.

```
reply(text)
```

Replies to the message, returns a Promise.

```
replyImage(imageData, text)
```

Replies to the message with image data and optionally text, returns a Promise.

```
replyImageId(imageId, text)
```

Allows you to re-use an already uploaded image.

```
getCommand()
```

Returns the command that was called.

```
getSender()
```

Object describing the sender of the message

D.4.2 Conversation

Represents a conversation

```
send(text)
```

Sends text to a conversation, returns a Promise.

```
sendImage(imageData, text)
```

Sends an image to a conversation, returns a Promise.

D.4.3 Utils

You can use the utils property of your plugin to get access to the utilities below.
findConversation(name)

Returns a conversation by its name, or throws if it can’t be found.

allGroupConversations()

Returns all group conversations.

allUsers()

Returns all user ids that Nacho has open chats with.

sendMessageToId(id, text)

Sends text to a user by their user ID, returns Promise.

sendMessageToEmail(email, text)

Sends text to a user by their email address, returns Promise.

emoji(shortcode)

Returns the unicode symbol for an emoji based on a shortcode, see http://www.emoji-cheat-sheet.com/for a list of shortcodes.

uploadImage(imageData)

Uploads image data, returns a Promise that resolves with the image id.

D.5 Getting your plugin running on the actual Nacho

Having a working plugin is great, but the goal is having it run on the actual main Nacho installation. To do this, you need to get your plugin to me somehow.

If it’s a one-off plugin, you can do this by just sending me an archive with your plugin in it, but you can also ask me to setup a repo on github to host your plugin, so you and others can work on it.

D.6 On natural language processing

You may know that Nacho also has some rudimentary NLP. I have not included that part of the codebase with the distribution above.

If you want to have your plugin triggered by natural language, talk to me after it is running within the main Nacho instance.