Team Documentation

Introduction

This document's purpose is to give a brief look into the way the team structured itself over the course of this project. This enables other teams to learn from practices utilized in this project, make them work together more effectively and avoid mistakes of the past. Furthermore, a continuous documentation of the team's current state may also expose potential short comings in team organization, as one is forced to permanently monitor how the team is interacting.

Process model

The process model which was chosen for this project is Scrum[1]. Scrum is based on being in close contact with the client, it fits perfectly to our schedule which envisages meeting the client once a week. Moreover, Scrum focuses heavily on the team's moral as a key factor in the project's success. A student project should be fun in order to make it easier for the participants to digest the new material they are confronted with. Srum provides for that need.

Due to the small size of our project team and only one team member having worked with Scrum before, we made the rather unique decision to combine 'Product Owner' and 'Scrum Master'. Attention, this is not good practice[2].

Tools

To assist us in implementing Scrum, we mainly employed *GitLab*'s sprint planning feature. *GitLab* was also used for version management and revision. Communication was done via e-mail and *WhatsApp*. For creating documents, we mainly utilized *Google Docs*.

Roles outside of Scrum

Apart from the 'Product Owner' and the 'Scrum Master', we decided that it makes sense to have one person of the team continuously working on the technical documentation and one on the team documentation. Experiences from old projects showed that documentation was often done poorly, because teams neglected its importance and only remembered it when deadlines approached. This leads to important decisions being long forgotten when the documentation is actually written and makes it hard for new teams to pick up the project. We wanted to steer against that by focusing on the documentation from the beginning.

The team documentation is crafted by the 'Scrum Master', so he always must actively reflect the team's dynamic.

---

[1] Further details: https://www.scrum.org/resources/what-is-scrum (Last checked 2020-10-26)

[2] Cf. https://en.wikipedia.org/wiki/Scrum_(software_development)#Product_owner (Last checked 2020-10-26)

Progress on documentation is shown every two weeks to emphasize its significance and move the current team's and project's state into focus. Providing an opportunity to discuss possible problems and hence become more efficient.

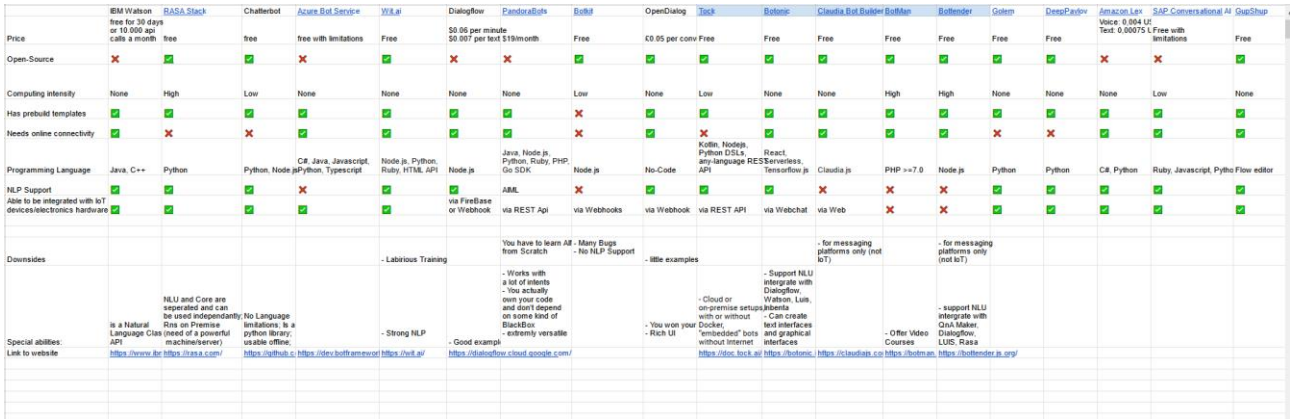
Additionally, we made the decision to designate a person which creates client meeting presentations in order to have one consistent visual theme.

Exemplary processes / decisions

One of the key decisions of this project was choosing the framework, we wanted to work with. This process was particularly difficult, since there was a wide range of different frameworks to choose from. So, researching this topic had to be done by multiple members of the team. Also, making the correct decision was crucial for the project's success, as it had a huge impact on following workloads. Choosing the wrong framework could have meant that the team would have had to implement features themselves which a better framework would have come included with. It even could have led to not being able to produce a working prototype if the right interfaces had been missing.

The way we tackled this problem was to first research possible criteria chatbot frameworks can be judged by. Next, we looked up which frameworks are out there. Then, we created a shared table including all criteria and frameworks.

We divided the frameworks into groups and assigned each group to a team member for research. The shared state of the table turned out to be crucial. It helped us to save time, because team members could still enhance the matrix, even if they found information about frameworks which were not part of their group. Finally, we asked the customer for requirements the framework must meet and decided on one framework together.



	IBM Watson	RASA Stack	Chatterbot	Azure Bot Service	Wit.ai	Dialogflow	PandoraBots	Botkit	OpenDialog	Tock	Botonic	Claudia Bot Builder	BotMan	Botfender	Golem	DeepPavlov	Amazon Lex	SAP Conversational AI	GupShup
Price	free for 30 days or 10.000 api calls a month	free	free	free with limitations	Free	$0.06 per minute $0.007 per text	$19/month	Free	£0.05 per conv	Free	Free	Free	Free	Free	Free	Free	Voice: 0.004 Lit Text: 0.00075 L	Free with limitations	Free
Open-Source	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓
Computing intensity	None	High	Low	None	None	None	None	Low	None	Low	None	None	High	High	None	None	None	Low	None
Has prebuild templates	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Needs online connectivity	✓	✗	✗	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	✗	✗	✓	✓	✓
Programming Language	Java, C++	Python	Python, Node.js	C#, Java, Javascript, Python, Typescript	Node.js, Python, Ruby, HTML, API	Node.js	Java, Node.js, Python, Ruby, PHP, Go SDK	Node.js	No-Code	Kotlin, Nodejs, Python DSLs, any-language REST API	React, Serverless, Tensorflow.js	Claudia.js	PHP >=7.0	Node.js	Python	Python	C#, Python	Ruby, Javascript, Pytho	Flow editor
NLP Support	✓	✓	✓	✗	✓	✓	AIML	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓
Able to be integrated with IoT devices/electronics hardware	✓	✓	✓	✓	✓	via FireBase or Webhook	via REST Api	via Webhooks	via Webhook	via REST API	via Webchat	via Web	✗	✗	✓	✓	✓	✓	✓
Downsides					- Labirious Training		You have to learn All from Scratch	- Many Bugs - No NLP Support	- little examples			- for messaging platforms only (not IoT)		- for messaging platforms only (not IoT)					
Special abilities:	is a Natural Language Clas API	NLU and Core are seperated and can be used independantly; Rns on Premise (need of a powerful machine/server)	No Language limitations; Is a python library; usable offline;		- Strong NLP	- Good exampl	- Works with a lot of intents - You actually own your code and don't depend on some kind of BlackBox - extremly versatile		- You won your - Rich UI	- Cloud or on-premise setups with or without Docker; "embedded" bots without Internet	- Support NLU intergrate with Dialogflow, Watson, Luis, Inbenta - Can create text interfaces and graphical interfaces		- Offer Video Courses	- support NLU integrate with QnA Maker, Dialogflow, LUIS, Rasa			
Link to website	https://www.ibr	https://rasa.com/	https://github.c	https://dev.botframewor	https://wit.ai/	https://dialogflow.cloud.google.com/				https://doc.tock.ai/	https://botonic.	https://claudiajs.co	https://botman.	https://botfender.js.org/					

Figure 1: Framework research table

Experiences

It turned out that handing the role of 'Product Owner' and 'Scrum Master' to one person worked. But this was only due to the fact that the team's supervision in the form of Mr. Schöler, Mr. Legat and Mr. Kottre possessed a lot of technical knowledge and acted as the 'Product Owner' often during meetings. Hence it is still advised not to follow this approach.

Also, the decision to have only one designated person which creates client meeting presentations was later revoked as it proofed to be too complicated to implement. The person, who researched the information, often also had the best vision to present it and the involvement of a designated presentation creator simply created unnecessary workloads.

Improvements

There were problems with the presentation of work at customer meetings.

The first problem arose since the constellation of people at customer meetings was not stable on the customer / team supervision side and the project team was not informed about the absence.

There were times where the project team wanted to discuss important questions concerning matters the university was responsible for, but the person representing the university was not attending the meeting. This meant that the project team had to switch to e-mail communication and hope they will receive a response in time or worse postpone the matter to the next week, hindering sprint planning.

On other occasions presentations were prepared, yet the person, the presentation was prepared for, did not attend the meeting. This cost time as team members had to reprepare the presentation for a later point of time. Also, it extended future meetings as the old presentation had to be crammed into meetings where new matters had already come into focus.

A solution for this would be to clearly communicate when, who attends the meeting.

The second problem concerned the presentation of running code at client meetings. Often, the program just did not work as expected. This was due to the instability of the environment the code was presented in.

A solution would have been to screen record the interactions, which the project team planned to demonstrate, in advance and simply replay them for the customer at the meeting.

Another point for possible improvements is the documents for the project work provided by the University of Applied Sciences Augsburg. In contrast to the product requirements, they were almost entirely in German. Fortunately, the scrum master of our team understood German. But if by coincidence all team members had been non-German speakers, this would have been an enormous problem.

Risk List

Risk	Probability	Impact	Total	Priority
Framework installation problems	10	10	100	high

Wrong decisions based on lack of knowledge concerning field	9	9	81	high
Team member sickness	8	8	64	high
Time planning	8	8	64	high
Insufficient Documentation	6	10	60	high
Language Barriers	10	6	60	high
Data loss	5	9	45	middle
Insufficient requirements	4	10	40	middle
Misunderstanding requirements	5	8	40	middle
Data leakage	3	10	30	low
Intercultural miscommunication	3	9	27	low
Hardware problems	2	7	14	low
Cost risk	2	6	12	low
Problems due to Covid-19	10	1	10	low

Risk	Points	Measures for risk minimization
Framework installation problems	100	- Framework set up on each PC as a group - Documenting any errors during that process
Wrong decisions based on lack of knowledge concerning field	81	- Extensive research phase
Team member sickness	64	- Weekly team meetings where individual progress is presented to make everybody able to take over each other's tasks
Time planning	64	- Short sprints - Adaptable requirements
Insufficient Documentation	60	- Weekly investigates of the current state of the documentation
Language Barriers	60	- Try to use English only for the entire project
Data loss	45	- Clear Git hierarchy
Insufficient requirements	40	- Weekly client meetings
Misunderstanding requirements	40	- Visualization of requirements
Data leakage	30	- Don't use real company data and structures for prototyping