# Masterarbeit

## Evaluation and Implementation of a motivational bot to support documentation process in mechatronics system design

vorgelegt von

cand. Bhargavi Mohan

Matr.-Nr. 6843331

Betreuer:

Prof. Dr.-Ing. Roman Dumitrescu
Prof. Dr. Eyke Hüllermeier                                    Paderborn, 31.05.2021

Masterarbeit Nr. MA0188

**Evaluation and Implementation of a motivational bot to support documentation process in mechatronics system design**

am: 31.05.2021

**FRAUNHOFER-INSTITUT FÜR ENTWURFSTECHNIK MECHATRONIK IEM**

Projektgruppe Entwurfstechnik

Zukunftsmeile 1

D-33102 Paderborn

**Sperrvermerk**


**Diese Arbeit bleibt aufgrund vertraulicher Daten und Informationen für die Öffentlichkeit gesperrt.**

<table>
<tr><td>_____</td><td>_____</td></tr>
<tr><td>Prof. Dr.-Ing. Roman Dumitrescu</td><td>Bhargavi Mohan</td></tr>
</table>

**Eidesstattliche Erklärung:**

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Paderborn, 31.05.2021

**Abstract**

The product development process in mechatronics implies intensified cross-disciplinary interactions. This in-turn results in an increased information exchange among them. Information in terms of artefacts, strategies and decisions are necessary to be known by all the teams involved in product development. The decision making is the central process of mechatronics specially design decision making that is part of the majority of the development process. The designers, developers and other stakeholders would need the right amount of knowledge transfer across different domains to deliver successful products and systems. One of the ways in which knowledge flow can happen is through having good documentation practices in each of the team. This helps both, members of different teams and new members of the same team. Although, the process of documentation is a long and an unexciting task, studies have proven the value of good documentation as well as implications of insufficient documentation. Having known the worth of documentation, it is a good idea to bring about a motivational factor to the task of documentation. On the other hand, since knowledge/information sharing has great benefits in mechatronics, collaboration tools are mandatory. These collaboration tools also contain information that would be worthy of documenting, but the useful data might be present in an unstructured way, challenging to figure out the essence of a particular conversational stream. Meanwhile, the usage of software bots in the collaboration tools help in expanding performance of an individual or a group of individuals. In this thesis report, a software bot is viewed as a motivational factor to support documentation of information such as design decisions that are discussed over collaboration tools. This report is an investigation of how a software bot can be realized to perform documentation of important design oriented decisions with minimal human interference. The bot presents the users with several suitable features for minimizing the efforts of documentation and maximizing the knowledge exchange activity in mechatronics.

# Contents                                                              Page

# List of Figures

# **List of Tables**

# Listings

# 1    Introduction

Mechatronics is an engineering field that comprises of mainly mechanical, electrical and computer science domains [Neu]. Mechatronic product development means the independent development of mechanical, electrical and software parts that are later united to constitute the entire system. The process of development of such a system has numerous challenges [BVK$^+$16] [ACFT$^+$]. These challenges are mainly due to the presence of a wide range of disciplines and stakeholders. To overcome these challenges and to promote collaborative, simultaneous engineering, Mcharek et al. propose a useful framework called "Knowledge Based Engineering" in [MAH$^+$]. This helps the engineers in one of the main tasks — decision making [WE][IR]. The study of mechatronics can be split into a variety of other focus areas like sensors, actuators, system modeling, logic systems etc. It involves complex decision making throughout the life cycle of product development process[HJC$^+$]. Therefore, mechatronics product development happens to be an extremely interactive domain. Due to this multi-disciplinary nature, the team members involved in mechatronics system design also come from multiple fields of engineering. This demands a high level of collaboration among the team members for a successful system design or product development [Neu]. Collaboration tools become very significant in such a team for all it's members to be aligned. Tools like Slack [LZSS] and Microsoft teams [HB] are extremely popular in the developer community. Additionally, such tools also aid the exchange of knowledge, information, updates and decisions about the development during the process. Furthermore, collaboration tools these days are highly sophisticated with numerous useful bots[LZFS]. Bots are nothing but an interactive interface for users to avail software services. They can also be termed as artificially intelligent, capable of comprehending commands from users or sometimes smart enough to carry out automated tasks. These bots are said to be highly convenient to carry out regular tasks. Conversational bots these days are so engaging and intelligent that it is often difficult to spot differences between a human and a bot[MP]. There are also software bots that aid a variety of tasks related to code development, code maintenance and other productivity tasks through simple commands. Further, Carlene et al. highlights that these chat bots help reduce the collaboration friction among the software developers[LZS]. So far, some of the fundamental responsibilities of a mechatronics team such as design, decision making and collaboration through the product development process has been discussed. Going ahead, another salient task is highlighted — documentation. There are numerous benefits attributed to the documentation activity. For

instance, up-to-date knowledge is recorded and as a result, all the learning and know-hows is conserved. The documentation also serves as a road map for the further developmental activities. Lack of documentation leads to lack of maintenance, lack of complete, updated systems and lack of traceability of the crucial design decisions. Therefore, Heesch et al.in [HAT] discusses why decision documentation can be valuable for the junior engineers.

In this master thesis, the mechatronics systems/mechatronic product development process is being studied, different types of challenges in the product development process is assessed, with that knowledge relevant problems pertaining to documentation of design decisions is analyzed, how collaboration tools can act as a beneficial medium to gather key design decisions is explored and finally a bot that can help transform that data as a document for future reference is prototyped. To begin with, there will be initial screening of some of the existing bots to see if they can be adopted to help achieve the goal of this thesis in the context of documenting design decisions.

## 1.1      Problem description

Mechatronics being a diverse team raises certain concerns in the course of product development life cycle [BVK$^+$16][Neu]. A few pertinent issues are studied in this masters thesis. Based on the literature review, there are two most important issues that needs to be addressed in a typical mechatronics team. These two are separate issues and are yet inter-dependent. The first problem causes the second problem below. So, problem number 2 is the major issue of concern.

1. Among the multiple domains within a team, there is room for improvement to exchange knowledge with respect to terminologies, functional requirements, design parameters and developmental processes. Weak knowledge management can lead to confusions within team members and hamper the design and development. There needs to be an organized method for the required knowledge sharing task. Furthermore, as mentioned above, mechatronics product development implies intricate details about design, architecture, implementation decisions and once again documenting(knowledge management) such important decisions become significant [Neu]. Documentation in turn promote knowledge exchange activity.

2. The major issue is lack of care for explicit documentation, particularly decision documentation. From the problem description number 1, the inference is that poor documentation is a consequence of poor knowledge management. Therefore, the

focus here is to study the decision making challenges[HJC$^+$] and going further, the need to document all the decisions. Decision documentation is a process of capturing vital information in a readable format, often required for future references. The authors in [MTA$^+$] highlight the fact that documentation of architectural decisions in the industry are usually neglected. The cost of poor or no documentation in the field of systems engineering can be well comprehended from [Kas]. Therefore, there is a clear purpose for documentation in mechatronics product development. Since collaboration tools are a critical medium for co-ordination among mechatronics team members, most of such decisions lie inside the collaboration tools. They are often present in the chat logs or sometimes even lost during the online meetings. How to differentiate between productive and unproductive material in chats/group chats [ZC]? How to extract any pivotal design decision that was made on a collaboration tool? For instance, on call whiteboards are increasingly getting popular lately, how does the team manage to capture any architectural/design decision that could be useful at a later stage [GW19]? Besides, going through long threads of chats for any vital data or decision will be a tedious task. How to make this process hassle-free? Henceforth, there is need for motivation to manage knowledge and documentation in such a team and also have a more structured mechanism to tackle the above issues. There have been several tools as part of modeling platforms in order to record the design decisions[MTA$^+$] where as there are no means yet to secure the design decisions that are present in collaboration tools.

## 1.2       Objectives

The primary objective of this master thesis is to aid employee motivation specifically with a focus to uphold design decisions documentation. The ultimate objective is to prototype a software bot that promotes employee motivation in the same focus area. Lately, software bots are useful in numerous fields like engineering[SSR$^+$], development [ABS] and HR [Moh19]. This type of employee dependence on such software bots bespeaks that the bots are generally helpful in motivating employees in one way or the other in order to reach their milestones. Research carried out by Suri et al. [SEH] say that there is usually a business case developed for the usage of software bots in organizations. The research summarizes the factors that are critically important for development of such a business case in which employee motivation constitutes 71%. Thus, it can be deduced that software bots boost employee motivation in general. Therefore, in this thesis, employee motivation to documentation is achieved through design and development of a software bot but, considering the scope of the thesis, motivation to document design decisions is the main

focus. The objectives of the bot are intended to drive the design of a software bot and help
decide on the features to be implemented. The software bot will have one major objective
and 2 optional objectives as follows. These in turn reflects the features of the bot that is
discussed in the Section 1.3.

1. This is the major objective and holds a higher value in priority of execution. The goal
   is to find an optimal solution to trace the design decisions present in collaboration
   tools. The employees use these tools for regular communication and co-ordination
   [LZSS]. Many prominent decisions would be made over these tools in group chats.
   There could be design decisions, organizational decisions, management related
   decisions. They can all be present jointly in a single chat. Later on, it is not a simple
   task to go through them to access any desired message that indicated a design related
   decision [ZC]. Hence, the aim is to mark the relevant messages using a bot in a chat
   during desired conversation so as to make them handy later when required [GW19].

Next up are the two optional objectives that hold a lesser value in priority of execution. The
execution of these depend on the time remaining after a successful execution of the major
objective explained above. Out of the below two objectives, the former one solves the
possible inconsistency in knowledge management that can occur in an inter-disciplinary
team. The latter one talks about a more general employee motivation with respect to mutual
acknowledgment in a team like mechatronics.

1. This optional objective is to improve knowledge exchange spanning across multiple
   teams. Lack of knowledge exchange hampers the collaboration thus leading to poor
   motivation among the employees [Neu]. There are knowledge about design models
   from different disciplines and it is challenging to evaluate all of them in parallel
   [ACFT$^+$]. Huge amounts of complex information or knowledge sharing would help
   the team to align better during product development process. This approach will also
   help the employees to obtain a holistic perspective on a product development which
   in turn adds up to employee motivation. Nevertheless, a lot of collaboration tools
   [HB] are available for this purpose but the idea here is to use these tools to further
   exchange information in an effective and an efficient manner with the help of a bot.

2. The next optional objective is concerning employee recognition. It is said that
   employee motivation enhances both an individual's performance as well as an
   organization's performance [MBP]. These recognition could be managerial-based
   recognition or coworker-based recognition. This helps employees to develop a sense
   of positive competition among themselves which directly influences the company's

growth. In this case, employee recognition takes place in collaboration tools and there are already a couple of software bots like HeyTaco[1] and Achievers[2] that help to encourage employees to complete their assignments. Once again, the purpose of this objective is to encourage positivity and motivation among employees using a bot.

## 1.3      Solution idea

The features of the motivational bot are listed in this section but like mentioned above, there is a *key* feature that is prioritized and is implemented first. Although, a software bot that is capable of rendering all three features is desired for a mechatronics team, the preference is given to the major objective that drives the *key* feature as follows.

The proposed solution in this work is to device an intelligent bot that supports employee motivation mainly in the process of documentation in a team like mechatronics. Documentation is even further narrowed down to design decision documentation. The initial step for this is to firstly evaluate the existing bots, i.e., to carry out a research to see if there are bots already capable of fulfilling the major objective stated in 1.2. If there are bots that are found to suit the requirements, the work will be towards adapting it's functionalities to prototype a bot that is exclusive to mechatronics domain. If there are no such existing bots, design and prototype of an appropriate bot that solves the major issue pointed out in 1.1 will follow. This bot will be implemented to be added in Microsoft Teams.

- The *key* feature of this bot is to track all the important design decisions that are present in an unorganized fashion inside a channel of Microsoft Teams. The bot should provide suitable means to differentiate the messages between general discussions and design related decisions in the group chats and present either a dashboard or a PDF file in a readable format. The dashboard and the file would display a summary of all the discussed design decisions, decision makers, date of decision and type of decisions. The user should be allowed to save the file to their local system. The default file name can be the combination of the channel name and a term specific to mechatronics design phase so that, it reflects the name of a popular mechatronics terminology that is universally perceived. This way, it is easier for a user to refer to it at a later stage.

---

[1] https://www.heytaco.chat/
[2] https://www.achievers.com/gb/

The below features may or may not be implemented in this work. If the *key* feature is successful well before the deadline, the following optional features will be implemented otherwise, this can be considered as part of future work. However, it is good to discuss ideas for now or later.

- This feature is to tackle the issue of lack of knowledge exchange in a mechatronics team. The bot can act as a translator for terminologies, knowledge base in terms of data, information, requirement parameters and development processes thus helping the knowledge flow across the inter-disciplinary teams. This may also be termed as a center for any FAQs. For instance — A user from a certain domain A should be able to interact with the bot by querying, through simple commands and receive the desired information pertaining to a different domain B.

- The next feature is to motivate employees by acknowledging their work and efforts in any project. The bot is designed to appreciate the team members who have worked to successfully attempt an innovative design strategy with respect to architecture and design or members who made substantial git commits and have closed high priority JIRA[3] stories/issues/tasks with respect to development or members who discovered a path-breaking strategy with respect to project management. The bot grants reward points for every task completed in their own focus area. For instance — in the area of software development, the reward points depend on tasks completed versus bugs returned before a sprint is over. After a certain number of points have been achieved, the employee could be made eligible for a more valuable reward. Certain gamification ideas can be used in this case, the messages can appear like in a video game. For example, after every 5 tasks successfully closed(without any bugs returned), the bot could posts a motivating message like — "Hurray! Level 1 completed! It is now time for you to accomplish next challenging level" and so on.

## 1.4     Thesis outline

This section describes how rest of the thesis report is organized.

Chapter 2 gives a brief overview of the fundamental concepts required to understand further chapters in the report. This helps the readers to know a bit of history of mechatronics, process of decision making and general idea about collaboration tools and software bots.

---

[3]https://www.atlassian.com/software/jira

Chapter 3 summarizes the literature survey that was carried out during the thesis tenure. This survey describes the avenue taken to understand the discipline of mechatronics, it's product development, decision making procedures, challenges in mechatronics domain and best practices. The survey then takes a shift to understand collaboration tools, it's advantages and disadvantages and finally the study of software bots and how they can be made use to complete tasks in an easier way.

Chapter 4 describes the research task of this thesis topic i.e. Evaluating the existing bots to verify if there is already a functionality in place to tackle the problem description. If yes, how should it be upgraded to develop a better version. If not, then the development of a bot pertaining to the objective follows.

Chapter 5 shows how the bot is designed. The architecture, workflow and technologies used for the design are discussed in detail. Chapter 6 talks about the implementation aspects of the bot. Sequence diagrams, code snippets and the screenshots of the working bot are presented to help the readers understand the technicalities of the bot.

Chapter 7 is the validation of the developed bot. It proves the bot that has been developed fulfills the requirements and objectives stated.

Chapter 8 concludes the report by suggesting some of the pointers for future enhancements.

## 2        Background

### 2.1        Mechatronics

The origin of Mechatronics engineering dates back to the end of 1960. The term was coined by a Japanese engineer "**Tetsura Mori**"[DHD17]. It is only during the 1980's that the domain acquired distinction and is now well accustomed. The advantages of the mechatronics systems has obtained wide acceptance all over the globe. This domain has over 10 technical sub-domains within itself, in the area of its R&D [bri10]. Therefore, the study of mechatronics systems is an amalgamation of various other technological and engineering subjects. It is serving as a gateway to massive scientific and technical studies. One notable observation is that the study of mechatronics is not only the combination of other engineering domains but also includes non-engineering domains [Bra04][Haba]. Therefore, it recognizes the possibility of advanced ideation and conception.

The interdisciplinary nature of mechatronics is an essential viewpoint. Studies show that the technological advancements can not be distinguished as per the formal disciplines. The next-gen systems could probably be formulated from the inter-working of multiple unrelated disciplines of operations in science and technology[WTH]. This science paradigm is also considered to have a philosophical side that motivates creative thinking, initiation of modern designs and modeling smart systems. The objective here is to explore advanced technological prospects by merging diverse domains together in order to produce new, intelligent techniques and systems approaches. The end product is manifested in a way that it is characterized with topnotch performance, accuracy, dependability, resilience and reasonable qualities. On the whole, the synergy in mechatronics systems design and development empowers great efficiency in terms of improved functionality, high reliability, better quality, effective scalability. Additionally, it is heading towards shortening developmental life cycle time and managing economic friendly methodologies in its evolution. Hence helping to provide simpler solutions to complex problems[Habb].

### 2.2        Collaboration tools

Collaboration in this context is defined as the activity of discussing task related topics among different members of a team. Tool in this context is a software application that assists a particular task[LBP]. Thus, collaboration tool supports communication on

different levels among members. Some of the categories of collaboration tools are version-control systems, ticketing systems, build tools, knowledge tools and communication tools.The main focus in the thesis is communication tools. Some of the well known communication tools are emails, forum websites or software applications like Skype, MS teams, slack, Google Hangouts Meet etc.

Communication among members of any team is made simple and easy by the use of the online tools. The team members can be situated in any part of the world to use collaborative tools irrespective of the geographical limitation. They can all come together to discuss their topic of interest through the collaboration tools. Fundamentally, the nature of a combined team of mechanical, electrical, software, in other words mechatronics, is that their foremost task is co-ordination among the team members. Collaboration tools act as enablers for them to discuss problems, solutions, decisions and ideas among other team members. Furthermore, they also aid knowledge sharing, information and project management[LBP]. In this thesis, slack and MS teams are the two main tools that is inspected. More information on these tools are discussed in the following sections.

## 2.3        Decision making in mechatronics

The process of a good decision making is crucial in a team like mechatronics because it is the one that decides how other technical processes as well as management processes should be carried out. A decision, especially a design decision should be carefully examined to study its repercussions that it could have on the system later. The design decisions should be made in the appropriate time because the premature design decisions could have serious impacts on system, cost and throughout the entire life cycle. Further, there will be decision gates defined that makes sure to team members that all the earlier tasks are up-to-date and are accomplished before getting along to the next set of tasks. Decision gates are also viewed as "milestones" or "reviews". The incoming and outgoing criteria are designated for every gate during the project management timeline. It is during this time in the project life cycle, a task is validated and approved by the project head[WE].

The design decisions are crucial in Architecture definition process, Design/concept definition process as well as Implementation process. In each of the process, there are three or more stages where different types of design decisions are made. In the architectural definition process, the types of design decisions are related to functional, logical and physical elements. In the design/concept definition process, the design decisions are stressed upon design definitions, design characteristics, design enablers and

design alternatives. Besides, in the implementation process, the design decisions are now more concrete hence the domain specific design decisions become important. Thus, mechanical, electrical and software design decisions are taken care during the process.

The authors in [IR] highlight that the evaluation of design decisions across the initial goals is vital during the product life cycle. There is another significant step that authors stress i.e, documenting the design decisions during the developmental process. They say that it will help to review future decisions. In the final phase of the product life cycle, a decision analysis process is carried out in order to quantify and asses the current data and suggest further enhancements. This way, it helps to decide on the best "design solution". There are many decision analysis techniques established to ease the analysis process. Interested readers can take a look at [IR] to view the name, purpose, timeline , results of the analysis process and much more to understand more about the decision analysis techniques.

## 2.4      Software bots

Bots/Software bots are nothing but an interactive interface for users to avail software services. They can also be termed as artificially intelligent, capable of comprehending commands from users or sometimes smart enough to carry out automated tasks. Collaboration tools like slack and MS teams contain software bots. Developing and hosting simple bots are pretty simple with the latest documentation available in online sources. In fact, Microsoft and Facebook also provide developers with necessary bot frameworks, toolkit and other forms of support. Bots these days are capable of assisting the users with almost every task in their routine[LSZ]. For example — reminders, support with FAQs, development, presentation and many more. In this regard, the biggest application of a bot in mechatronics is that it aligns the multi disciplinarian team members in terms of design and development process, decision making process etc[SSR$^+$].

Each day, there are novel developments in the research and development of bots. Developers are trying to build such bots that it is capable of supporting the users with natural-language interpretation techniques. Due to huge advancements in the field of bot development, there is a need to clearly understand what is or is not a bot and hence the authors of [LZFS] have presented a taxonomy of software bots. The taxonomy is based on environment dimension, intrinsic dimension and interaction dimension. This makes it very clear about the properties, behaviors and the environment where the bots are being operated and designed. This helps the community to better understand, evaluate the

existing bots and also how can the future bots be made more innovative and constructive in their design.

## 3        Literature survey

This chapter is about the literature review carried out for this thesis. The aim of this section is to explain in what order the scientific papers were studied. The outcomes of referring the following papers helped to re-define the problem description and comprehend what is lacking and what are the apparent requirements clearly. It also aided to emphasize the apparent objectives and played a great role in establishing a holistic view of the entire thesis topic.

First, the plan was to simply understand how product development takes place in a mechatronics environment and the authors in [Neu] explore the strengths and weaknesses of Mechatronic Product Development(MPD). Due to the complexity of the multi-disciplinary domains, all the challenges with respect to the terminologies are exhibited. It presents the terminological clarifications that is required in MPD. One other main point here is knowledge sharing among the different domains. The paper highlights about benefits of this cross platform product development in terms of cost and features. It also talks about the drawbacks of the same in terms of complexity, development process and structure of MDP. Subsequently, the industry process of product development can be well understood in [BVK+16]. The paper explains in detail about Product Lifecycle Management (PLM), state-of-the-art challenges faced by PLM, the authors have proposed an approach for a better methodology and application of PLM process. The authors quote an industrial example, of a client case and hence have helped evolve PLM considering the current challenges. The authors of [ACFT+] in contrast reviews challenges in design of mechatronics systems. These challenges are mainly in the phase of integration of designs from different domains.  The authors stress the need to consider the complex inter dependencies between subsystems.   Further studies regarding the challenges of development in mechatronics led to [Thr]. The fact that the already existing mechanical and electrical systems are outdated. These legacy systems have to be replaced by updated systems where the functionality is taken care by software implementation. The authors also discuss the challenges in the development of mechatronic systems and a mechatronic component is also proposed. The main aim of this component is to aid those identified challenges in terms of integration, complexity, size. Further more, the survey journey had to associate the challenges of mechatronics to the next chief topic i.e. collaboration. The paper [MAH+] serves as the link and rightly explains the challenges and the need for collaboration. This paper briefs the importance of collaboration and knowledge sharing

among different disciplines. Knowledge based engineering method is presented which is adapted to mechatronic design. This method gives a way to help engineers to achieve collaboration therefore reducing design time and cost.

The survey now focuses on collaboration tools. Microsoft Teams and Slack tools were given most attention. Hence the book [HB] is regarding everything about Microsoft Teams and its features. It includes explanation about its characteristics and how it can be operated. It includes some background study that is its history with the application Skype[1]. The book gives all the essentials of this collaboration tool like creating teams, channels, scheduling meetings, automation of some processes with the help of teams, third party add-on like bots etc. Similarly, the popularity of Slack tool is well detailed in [LZSS]. It states how developers have been using Slack as their effective and efficient tool for collaboration during software development. The extensive use of bots, and the possible integration of other services is a major aspect why slack is so renowned in the developer community. This study is to mainly understand the significance of Slack that supports various processes of software engineering. The scientific work in [ZC] identifies how can common messages be distinguished by noteworthy messages in collaboration tools. This paper talks about ways to distinguish important and unimportant messages in chat logs. They emphasize the need for group chat analysis towards making it more structured. The tool "Tilda" helps in summarizing chats for the users using simple commands. The authors evaluate the tool against google docs[2] and different forms of notes taking and discuss the efficiency of the tool implemented. Hence, it is clear that the distinction and documentation of noteworthy messages, or a display of summary of all those messages in the collaboration tools would be a coveted solution for the problem description in the thesis. The type of messages that needs to be captured for later use is design decisions as per the requirement. This led to the next chief literature survey topic i.e. decision making process to understand the internal details of the process.

Initially, the study of decision methods had to be made. So, two well known books were referred to get a thorough and comprehensive understanding of decision making procedures in mechatronics. In [WE] the decision management has been explained in detail. It states decision situations in every phase of the lifecycle. It describes the key elements of a decision management process like input and output variables and the process activities. The process activities include defining decision management strategy, listing the differences when an alternate decision is chosen, analyzing and managing the decisions.

---

[1]https://www.skype.com/en/
[2]https://docs.google.com/document/u/0/

It also talks about alternative decisions through deterministic analysis and improving alternatives. Likewise, [IR] gives an overview of decision analysis process. It tabulates different review process in different phases of the product development. It also lists the purpose, timing and results of the review process. This paper highlights the fact that these review processes are helpful in the evaluation of technicalities , expenses and alternatives. It summarizes the evaluation methods and tools for the purpose of assessing a decision that was taken. Evaluation methods such as simulations, weighted trade off matrices, surveys, user review and comment and a few more can be used. In addition, the study of tools or systems that support the decision systems was interesting hence papers that give a good synopsis about decision support systems were examined. The work in [ZH] says the Decision Support Systems (DSS) have been well investigated over 40 years and further talks about the development of DSS applications. The authors review the progress of DSS research and examine Model-driven DSS, Data-driven DSS, Group Communication-driven DSS, Document-driven DSS, Knowledge-Driven DSS, Web-based DSS and finally explain the future of DSS. In much the same way, [Her] demonstrates the role of DSS in sustainable decision making. General principles of sustainable decision making is presented. Also, some examples where DSS can be applied to sustainable decision making are quoted. Authors have also discussed approaches to decision making and decision support systems. Finally, the academic work proposed in [HJC$^{+}$] signifies the need of inter-dependent decision making in the design phase of complex systems. The authors examine this situation by conducting interviews and work-shop with system architects. This helped them understand what decision support tools are required in the system architecture process. The gathered data from architects later helped authors to develop something called decision support Framework. This framework has 7 different decision domains related to system architecture life cycle. At this point, it became clear that a system/device/mechanism to support design and architectural decisions in a mechatronics field is the core requisite. Since communication is utmost important among all disciplines, the system/device/mechanism to support design decisions is best to be equipped inside collaboration tool in the form of a BOT.

Finally, the software bots were needed to be studied. The authors in [LSZ] depicts the advancement of software bots and their extensive usage. This paper gives an overview of how software development and bots go hand-in-hand. The paper gives a short description about creating and hosting bots. This paper also classifies the bots as collaboration bots, productivity bots, documentation bots etc. The paper lists some common bot platforms along with other bot technologies. Next, the authors of [LZFS] have presented a taxonomy of software bots. It is based on environment dimension, intrinsic dimension and interaction

dimension. This makes it very clear about the properties, behaviors and the environment where the bots are being operated and designed. Also, [LZS] examines how chat bots help reduce collaboration friction points in software development. The authors classify and present 3 friction points. The authors present a sequence of research questions that forms a gateway to future discussions on bots that can assist team collaboration. Besides, [MP] is a study that is specific to one bot — Replika. The main aim of this experiment was to focus on the anthropomorphic view during the interaction. After this experiment, the participants were asked how could Replika be more engaging. Lastly, a seminar paper [SSR$^+$] in which the authors brought together researchers and practitioners with diverse backgrounds. There were also questions raised on how to effectively design and use bots. There are definitions and outlook of bots, level of acceptance of bots by the developers, use cases of bots, inclusion of diverse community with the help of bots, bots that promote collaboration.

# 4        Evaluation of existing bots

The first step towards assisting orchestration of decisions(the term decisions always mean design decisions) of a mechatronics environment in collaboration tool is to screen the existing bots to check if there are any bots currently, that are capable of resolving the problem description as discussed in Section 1.1. The collaboration tools under this investigation are SLACK and MS Teams because of their global acceptance and usage. The bots available in these two tools would be under scrutiny. The end purpose of this phase is to examine the functionality and features of the available bots that exclusively handle topics like employee motivation(preferably towards documentation), decision management, decision documentation, message tagging/labeling, decision summary display or decision summary download.

## 4.1        Requirements description

This thesis has two segments, 1). Research and 2). Implementation. The research involves identifying the bots that fulfills objectives and resolves the problem description of this thesis. The implementation segment depends on the findings of the research and is discussed in the further sections. An immediate step in the research segment is to enumerate the **Requirement characteristics** for further examination and implementation. The existing bots are checked to see if they poses one or more Requirement characteristics that are listed below. If there already exists a bot with all the Requirement characteristics as it's features, then the bot should be improved by adding new features and functionalities i.e the optional solution ideas as discussed in Section 1.3. As a final step, if there as no bots at present that suit the below list, a new bot is implemented that meets all the Requirement characteristics. One implicit and obvious Requirement Characteristic of this thesis is "employee motivation" towards documenting vital decisions. Hence, each of the below characteristic has been carefully formulated to finally achieve this bigger objective.

1. The bot should be operational within MS teams(although SLACK bots are also studied, it is only to check if there exists a better feature that can be adopted to build a similar bot inside MS Teams).

2. The bot should be able to provide decision template with valid decision tags/labels. If the suggestions by the bot is specific to V-model design phases as per VDI standards, then it's a plus.

3. The bot should be capable of separating the normal conversations from the important design decisions in the chat window.

4. The bot should provide a functionality to view all the design decisions taken during a discussion by means of an associated tab/dashboard. In short, a decision summary should be offered.

5. The bot should be able to provision a download option. A copy of the decision summary should be downloadable in any readable format(.txt, .docx or .pdf)

6. The bot should also be capable of retrieving metadata(raw data) from the database. Working with raw data might be beneficial for some teams.

## 4.2 Existing bots and their features

In this section, a review of 3 bots each in Slack and MS Teams is presented to explain how they help the users in documenting design decisions. The strategy to select the below bots was — In the bot directory of the respective collaboration tools, topic related keywords such as "decision", "documentation", "summarization", "motivation" and "message tagging" were entered. The most applicable bots popped up. The bot description of all the bots were checked and the most relevant bots whose features were close enough to the Requirement characteristics qualified for the evaluation.

The keyword "motivation" fetched bots that assisted users in peer recognition and rewarding co-workers(which is also considered as optional objective). These bots are not evaluated for the time being due to it's lesser priority.

### 4.2.1 Slack bots

1. **tilda**[1] — A bot whose main feature is chat summarization. The main idea here is to group the incoming messages as questions, answers, ideas, information, action etc. The users are required to use the inbuilt commands to notify the bot what type of message are they exchanging in a group. For instance, the commands

---

[1]`https://tildachat.com/`

are /∼addquestion, /∼addanswer , /∼addinfo, /∼addanswer. These commands
are understood by the bot and it groups the user messages accordingly and gives
a summary upon using another command — /∼currentsummary. The summary
contains the people involved in the chat, the number of messages, date and the time.
The summaries are posted as messages by the bot in the same group and all the
members in the group can refer the summary up to a certain time. These summaries
may disappear later because slack offers to access messages only upto 10,000 recent
messages and the older messages can be retrieved by choosing a premium Slack
version. Therefore, these summaries need to be screenshot explicitly to save them to
the local system permanently.

2. **summarize bot**[2] — The feature of this bot is simplistic. It is only capable of
summarizing any public web link within a group chat or individual chat. The bot
can also summarize documents, images etc. The command used is /summarize
[url]. Keyword extraction, key fragments list, desired summary size, downloadable
results are its key features. The reason to review this bot was to understand the
summarization technique used in this bot to actually see how it differs from the
previous one.

3. **decisionbot**[3] — As the name suggests, this bot helps in quicker decision making
in a group conversation. This bot does separate the normal messages from decision
related messages and also stores them all in one place to some extent. A user can
use the decision making template by initiating the command— /decision and fill
the details such as title, description, decision maker, due date and submit the form.
A new channel with the decision title gets automatically created and the users are
allowed to edit the details in that channel. Other members can also be invited to join
the conversation. The "Make decision" option will also be available to the intended
person making the decision. Once the legitimate person submits the decision, the
bot posts the results of the decision on the channel and that channel is then archived.
However, the discussions around that specific decisions are brows-able even later
in that particular channel. Additionally, there is another tab in the bot channel
that shows a history of all decisions, a pictorial representation of open and closed
decisions over time.

---

[2]https://www.summarizebot.com/
[3]https://www.decisionbot.io/

### 4.2.2      MS Teams bots

MS teams does not have any bots to handle chat summarization, hence no bots that could help in documenting messages in chats. There were a few bots that could potentially isolate the important messages in a place within the tool and they are discussed below.

1. **we decide**[4] — This bot is very similar to **decisionbot** discussed in Section 4.2.1. This bot provides a decision making form to a user who can fill the title, description, assigned date, due date and assign it to one of the members in the group. The responsible person either approves or rejects the decision. Additionally, there are text-boxes to add action items in case of approval and rejection. Files can also be attached for any required reference. This bot segregates important messages via having separate decision forms and accumulates the messages via in one place using Microsoft tabs.

2. **perfony**[5] — This is also a bot that helps decision making in a team. It has a complex UI and features. The decision can be created with the assignee details. This decision form can be created in the form of a folder and can create further sub-folders for related decision items. There are 3 forms of views namely — folder view, kanbann view, Gnatt view. There are comment sections, attachment sections, bookmark option, download options, archive options, and also a status bar that can be dragged laterally to indicate the state of the current decision topic. The status bar indicates how much percentage of the decision making task is completed and how much is remaining. There are many more features that are not in interest of this thesis requirement hence are not listed here.

3. **approve simple**[6] — This bot was considered for the review because it claimed that it accelerated decision making in a corporate environment. Screening the the bot revealed that it was more useful for employees in the managerial roles and not much useful for decision making or documenting messages in group conversations. It is used as an approval tool for managers or directors in a 1:1 conversation with the bot. The user should type a command like *My Approvals* and the bot displays a queue of all the approvals in different contexts like leaves, time-sheet, budget etc. The manager can approve or reject stating the reason(optional).

---

[4]`https://appsource.microsoft.com/en-cy/product/office/wa200001566?tab=overview`
[5]`https://appsource.microsoft.com/en-us/product/office/wa104381418?tab=overview`
[6]`https://appsource.microsoft.com/en-nz/product/office/WA104381812?tab=Overview`

## 4.3          Evaluation criteria

This section tabulates the above introduced bots against the **Requirement characteristics** discussed in Section 4.1. The Requirement characteristics have been manifested to constitute the **"Evaluation Criteria"** to measure the appropriateness of the existing bots with regards to this thesis requirements as shown in Table 4-1. Every criterion is marked YES if the bot has a particular criterion as it's feature, it is marked NO if the bot does not have a criterion as it's feature and it is marked PARTIALLY YES if the bot somewhat meets a criterion but has a slight variation in it's functionality.

*Note: The evaluation criteria in the below table is introduced in the same order as the requirement characteristics in Section 4.1 so that it is easier for the readers to refer and relate each criterion with it's corresponding characteristic.*

| Evaluation criteria | tilda | summarize bot | decision bot | we decide | perfony | approve simple |
|---|---|---|---|---|---|---|
| Workspace - MS Teams | NO | NO | NO | YES | YES | YES |
| Message tagging/labeling(VDI specific) | PARTIALLY YES | NO | NO | NO | NO | NO |
| Design decision management | YES | NO | YES | YES | YES | NO |
| Design decision documentation | PARTIALLY YES | NO | YES | YES | PARTIALLY YES | NO |
| Design decision knowledge management | PARTIALLY YES | NO | PARTIALLY YES | NO | NO | NO |
| Easy metadata retrieval | NO | NO | NO | NO | NO | NO |

*Table 4-1:     Summary of evaluated bots against the evaluation criteria*

## 4.4          Discussion and Summary

Based on the evaluation of the existing bots, there is no bot currently available in MS teams that satisfies all the Requirement characteristics. There are a couple of decision management bots that also provide documentation of decisions but do not provide decision templates that are specific to V-model design phases rather a very general category of "decisions" documentation is possible. The knowledge management and data retrieval criteria is barely addressed by the existing bots.

There are some bots that require the users to explicitly pick a new decision template every time a decision has to be made. These templates are independent of the previous decisions and all the details like the current date, due date, decision name , type and decision maker's name has to be filled from scratch. If the users want to continue any discussions around one decision it is not possible inside the template. By choosing another template the conversation thread is lost. All these templates can not be posted inside a group but it will be present is a different tab away from the conversation window. The actual requirement is to have a template that can be embedded within the asynchronous group chat so that the

a thread can be maintained as well as there is clear distinction of common messages and messages coming from a pre-defined template, the quest was for such a capability in a bot.

Among the examined bots, "tilda" that is available in Slack seems to be the closest that has potential to tackle some of the requirement characteristics of this thesis topic. This bot has a general category "Decisions" that can be documented but it is not related to V-model specific decision topics. Another drawback of this bot is whenever a decision has to be made, a long command has to be typed and hence it is not a sound solution. The usage of these commands currently is too much action for the users. It can make the users exhausted and not want to use the bot. Hence, this difficulty should be reduced or eliminated.

Therefore, considering the week points in all the evaluated bots as well as the requirement characteristics that is necessary to reach the goals of this thesis, a new bot has to be developed. The bot should provide a template of predefined decision type. The date, name of the decision maker and decision names should all be auto-generated. In that way, the users only discuss the respective design decision topic without having to fill name and type of the decision. Additionally, bot should provide the best possible means to keep track of the previous decisions to be able to continue the thread within the chat window. This can be achieved by having the decisions posted by the bot inside the chat window. Furthermore, the bot has to make the usage of commands seamless and effortless. That way, the users are motivated to use the bot without spending too much time just on the commands. To do that, a messaging extension can be configured. Furthermore, a display of all the decisions discussed in chat window should also be present in the form of a summary. To do that, a separate tab can also be configured. A design and implementation of such a bot is presented in the following chapters.

# 5        Design details

The outcome of the research task previously has established clear signs for an outset of
a novel bot that enables all the *Evaluation Criteria* as it's features. This chapter is an
explanation of the design techniques involved in building the new bot. The readers will
be able to interpret the elementary idea and approach intended, concerned technologies,
high-level architecture and workflow of the bot. The bot is designed to operate in Microsoft
Teams. Further, it is designed in way that it can be added to multiple channels in the tool.
On the whole, this chapter gives the reader an idea of the architecture plan of D3driver and
a short technology review has also been done in the end.

## 5.1        Prerequisites

This section describes all the terminologies needed for further understanding.

- Microsoft Teams[1] — also called MS Teams is a well known IT collaboration tool
  used for communication purposes. It provides options for chat, group chats, audio
  and video calling, file storage and third-party apps integration.

- Microsoft Azure[2] — is a cloud computing platform owned by Microsoft. Users
  are offered a wide variety of cloud services like computing, developing, analytics,
  storage, networking, deploying their applications in the cloud.

- MS teams Adaptive cards[3] — Microsoft introduced the concept of cards to exchange
  the UI content in a consistent way. It is a simple JSON object that can rendered
  within a bot. There are 3 types of cards of which adaptive cards are used for
  implementing the new bot. These cards can be used in bots that can either be part of
  bot communication or the users are also able to exchange cards with other members.
  There are abundant formatting options for styling the text inside the card to make it
  look impactful.

- V-model — It is the widely known and is a highly accepted model for developmental
  process. A team like Mechatronics that involves diverse engineering groups to build a

---

[1] https://www.microsoft.com/en/microsoft-teams/group-chat-software
[2] https://docs.microsoft.com/en-us/azure/?product=featured
[3] https://adaptivecards.io/

system use this model as a guide in the different phases of development. It showcases the sequence of tasks to be carried out in each phase. The paper [GHB$^+$18] explains the state-of-the-art theories and concepts about the model.

## 5.2 Name and logo

The bot is responsible for driving the whole process of Design Decision Documentation hence it is named **D3driver**, short for Design Decision Documentation driver. Presenting the logo of D3driver in Figure 5-1. The bot is aimed at successfully gathering all the design decisions made during a conversation. Accordingly, the logo exemplifies a bot that holds all the important design decision documents on one side and some more collection of the same on the other side.
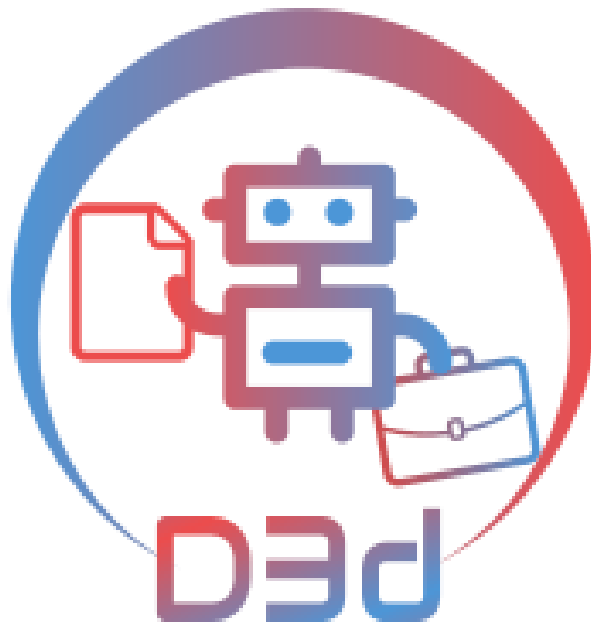


*Figure 5-1:     Logo of D3driver*

## 5.3 Idea and Approach

This section is to understand the conceptual perspective of D3driver. The methodology adopted to convert each of the evaluation criterion as a functionality/feature of D3driver can be identified as follows:

1. The first criterion is related to the platform of operation and D3driver is being designed and implemented in MS Teams by using all the technologies and applications that can function in MS Teams.

2. The second criterion is regarding message tagging/labeling with respect to V-model specific design phases. To this end, the bot considers the types of decisions in a mechatronics team that could possibly be discussed based on the V-model benchmark. A V-model serves as a standard for a team like mechatronics to carry out product design and development. Therefore, the bot design adheres to decision types put forward by VDI(German Association of Engineers) standards. The interpretations of V-models are explained in [GHB$^+$18]. These globally recognized and accepted design decisions form the decision template in D3driver. The templates are designed in way that messages are tagged/labeled according to decision names defined by VDI.

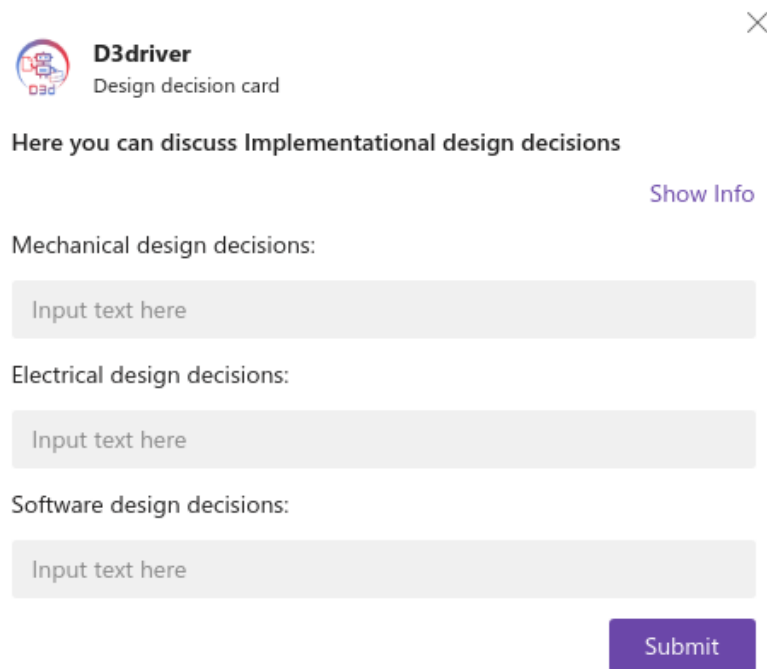3. The third criterion is the design decision management and to design this, the usage of adaptive cards in MS Teams is selected. By using adaptive cards, a clear difference can be made between common messages and important decisions.

4. The forth criterion is design decision documentation and for this purpose, the bot is designed to install a configurable tab when the bot is added to a channel. The tab is meant to display a summary of all the decisions discussed in the chat.

5. The fifth criterion is about knowledge management and D3driver will be designed to allow users to download a copy of the decision summary that is present in the configured tab.

6. The last criterion is about retrieving metadata from the database. D3driver will be also be designed to provide the raw data that is linked to a particular channel in the JSON format.

Furthermore, depending on the type of V-model, the design stage in a project may be extended across different phases in a V-model. Hence the design decisions will be discussed in all those phases and a provision to document all the decisions in all the phases is designed in D3driver. According to V-model process in an organization, the design decisions generally could be discussed in phases like 1) Architecture design 2) Design/Concept and 3) Implementation. There are different decision types in each of these phases with regards to design and hence each phase would correspond to it's own design decision type. In brief,

the approach here is to sort the design decision types based on the respective design phases
and they are as follows:

- Architecture design phase

    1. Functional design hence Functional design decisions

    2. Logical design hence Logical design decisions

    3. Physical design hence Physical design decisions

- Design/Concept phase

    1. Design definition hence Design definition decisions

    2. Design characteristics & Enablers hence Design characteristics & Enablers
       decisions

    3. Design Alternatives hence Design Alternatives decisions

- Implementation phase

    1. Mechanical design hence Mechanical design decisions

    2. Electrical design hence Electrical design decisions

    3. Software design hence Software design decisions

There will be three decision templates in D3driver. Each decision template is designed
based on one of the three V-model phases mentioned above. Every template will have three
of it's own decision types. A new concept is introduced i.e. ***Channel Initialization***. When
D3driver is added to a channel in MS Teams, that channel is supposed to be initialized with
one of the decision templates (V-model phase), which means the channel can document the
decision types that are part of the selected template (V-model phase). To do this, D3driver
is invoked using a specific command and is only done once during channel initialization.
After initialization, the templates can be accessed easily and are visible in the form of
adaptive cards. The bot is now ready to tag and document important decisions using
adaptive cards. An example of an adaptive card with the "Implementation phase" and it's
decision types can be viewed in Figure 5-2.

*Figure 5-2:     Example adaptive card of Implementation phase*

In addition to the above design techniques, the bot is designed to use as less commands as possible to overcome the disadvantages of continuous command usage(like pointed out during the research of existing bots). The evaluation of the existing bots has also shown that in the existing designs, the users are expected to type a long and a complicated command that is a combination of many ASCII characters or the users need to make more that 2 mouse clicks to start the prioritized discussions. D3driver is designed to make the routine more effortless and thus with the help of Microsoft's *Messaging Extensions*[4], the bot adds a static button adjacent to the other default buttons beneath the standard formatting text-box to access the decision templates. The D3driver can even be pinned to always stay visible and is just one mouse click away to start the prioritized discussions. A decision template (adaptive card) will pop up as shown in the 5-2 upon clicking the D3driver messaging extension and further there are dedicated text-boxes to discuss prioritized messages(design decisions) whereas the other conversations that has no priority happen normally using the standard text-box offered by MS Teams. After submitting a decision to the group, all of them are stored in the database for later extraction. The D3driver tab will have a display of all the design decisions and other details like the member who took the decision, type

---

[4]https://docs.microsoft.com/en-us/microsoftteams/platform/messaging-extensions/
    what-are-messaging-extensions

of decision and the date of decision. A more thorough explanation of D3driver's working
will be seen in the following chapter.

## 5.4            High-level architecture

The Figure 5-3 represents the high-level architecture of D3driver. The components of the
architecture diagram are as follows:

- **Users** — They symbolize the group members of a collaboration tool. The users
  converse with the other group members via the collaboration tool. They are the end
  users of D3driver.

- **Collaboration tool** — Microsoft Teams is picked as the collaboration tool for this
  implementation. This tool helps users to communicate with each other by means
  of allowing them to form teams and channels. This tool also has a rich set of
  conversational, productive bots that aid the users in their routine activities. This
  tool serves as a medium for sending and receiving messages to and from the users.
  Note: MS teams will come into picture once again after the bot has been hosted
  by Microsoft Azure. Teams is responsible for creating an important file along with
  logo icon-files to finally use the bot. This details of which will be discussed in the
  implementation chapter.

- **Channel connector** — A connection link between the Microsoft teams and D3driver
  is none other than the channel connector. To be able to deploy D3driver in the bot
  studio of MS teams, one must register it with Microsoft Azure bot service. Microsoft
  Azure has a set of approved channels. The developer must set the bot up with the
  appropriate channel available in Microsoft Azure. In this case, the bot should be
  configured with Microsoft Teams. This means the D3driver will be able to talk
  to users via Microsoft Teams. This connector is responsible for forwarding user's
  incoming messages to the bot's endpoint appended with the standard endpoint of MS
  Teams i.e. **/api/messages**. Here is an official documentation[5] on how to structure
  bots with channels with the help of Microsoft Azure

- **Bot Framework v4** — Azure Bot Service offers a complete toolkit required to build
  bots, for example the popular Bot framework SDK. This framework is the core of

---

[5]`https://docs.microsoft.com/en-us/azure/bot-service/bot-service-manage-channels?`
`view=azure-bot-service-4.0`
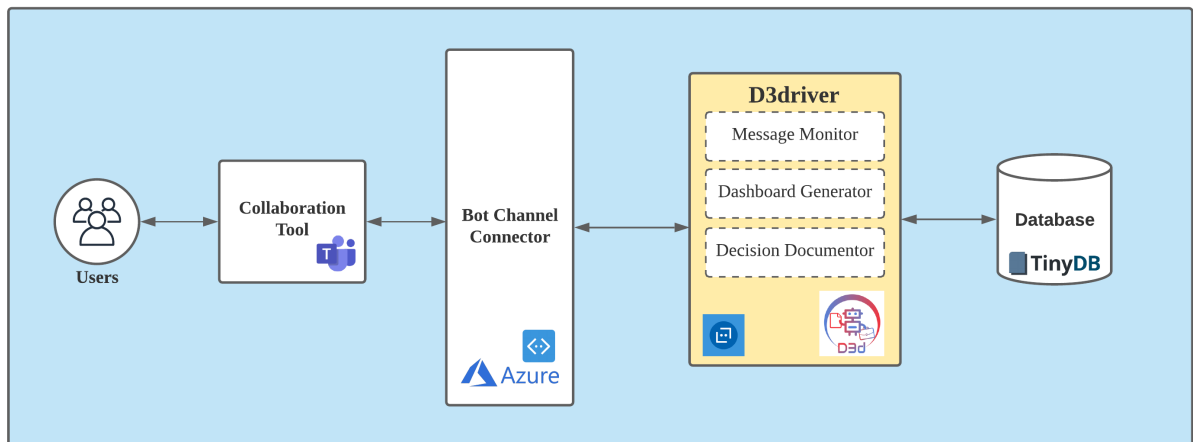
*Figure 5-3:     High-level bot architecture*

D3driver and is the main element for developing any MS Teams bot. The framework provides many schema and this implementation will use cards schema. Card schema is the representation of interactive adaptive cards in the application-level, that can be used inside a group chat. After the bot development, the developer is required to create a new bot service by providing all the desired data. Information like name of the bot, developer location, Bot template(v4), Bot ID and password etc should all be mentioned during the registration. This will help Azure to create the bot service and deploy the bot to the cloud. After successful deployment, the developer will prepare all the necessary code files to deploy. All the files should be zipped up i.e, the python files(app.py), dependency files(requirement.txt) etc. The deployment and the channel connector configurations should make the bot available inside MS teams bot studio. Once again the official documentation for this can be seen here[6]

- **Database** — This is the storage component in the architecture. This serves as the persistence layer for D3driver. The incoming decisions should be stored for later retrieval and hence it would require a simple database for that purpose. TinyDB is written in python and can be easily extended to make use of custom storage. The main function of the database is to store the details like channel name, sender's name, decision name and date etc. It can be devised to store any schema that a developer

---

[6]`https://docs.microsoft.com/en-us/azure/bot-service/abs-quickstart?view=`
`azure-bot-service-4.0`

would design. After storing, it can be queried to display any data the developer wants to display on the D3driver tab.

## 5.5        Work-flow of D3driver

This section describes the step-by-step working of the D3driver by means of a flowchart as shown in Figure 5-4. The flowcharts depicted has 3 parts. The left hand side of the flowchart denotes the main working procedure of the D3driver using messaging extension. The central part of the flowchart depicts the process of channel initialization and the right hand side of the flowchart depicts the operations that can be performed in the D3driver tab. As soon as D3driver is installed to a channel in MS Teams, two processes will take place. 1) D3driver's Messaging Extension will be installed and 2) D3driver's Configurable Tab will be installed. The channel initialization is done by invoking the D3driver. The bot will post a card where it asks the users to choose from one of the V-model phases. Any user selects a suitable V-model phase which is best according to their project requirements. After selection, appropriate messages will be posted by D3driver in the group as to which V-model phase has been selected and the name of the user who made the choice.

After a V-model process is selected, the D3driver's messaging extension button(referring to the left part of the flowchart) can be clicked to start a conversation related to design decisions. Upon clicking, an adaptive card(decision template) pops up with three other input fields and their names are dependent on the V-model phase chosen for the team/project during channel initialization. The user can make use of those intended fields to discuss design decisions. After a decision is typed, the users can click on the "submit button" within the card. Upon clicking the submit button, the typed decision is posted into the channel window. The typed decision also appears in the form of a card with name of the decision type, name of the decision maker, date of decision and the actual decision itself. This way, all design decisions are sorted from the other normal conversation material in the group chat. All these cards are saved into the D3driver's database. This step can be repeated until any given time unless the users want to discuss decisions that are a part of a different V-model phase, then the channel needs to be re-initialized as illustrated in the flow chart.

The right most part of the flowchart denotes the visualization of all the saved decisions. The purpose of having a D3driver tab is to display all the decisions in one place. After the decision cards were exchanged, all the information is saved to the database and is rendered as a HTML page on the D3driver tab. The work flow of this is also simple. Any user who

wants to take a look at all the decisions that has been discussed can click on the designated D3driver tab of a channel(There will be separate D3driver tabs for different channels and will contain the information that is regarded to the respective channel only). In the tab, a dashboard can be viewed with the type of channel initialization, decision type, date, decision maker etc. A search box functionality is designed to filter the desired decision card. A sort button functionality is designed to sort the decisions according to the date. A button that allows the users to download a copy of all the information present in the tab is designed. Finally, a button to view the metadata of that channel's decision cards is designed.

## 5.6        Technologies and Applications

The choice of technologies and applications that will be used for the implementation are listed in this section:

- Microsoft Teams — The D3driver app package is designed and developed in such a way that it can be integrated to any collaboration tool but this thesis implementation focuses on hosting the bot in Microsoft teams hence this app is the obvious choice.

- Microsoft Azure — This cloud service seems to be the best to work with Microsoft teams and also the platform ensures a great collection of bot framework features. The open-source bot framework SDK v4 is an ideal development toolkit for this implementation.

- ngrok[7] — is a tunneling tool that helps users to expose their local-host server to the internet. This is a free software that can be downloaded on the system. ngrok uses either the default HTTP port or an explicit port number can be mentioned such that ngrok exposes that local port.

- Python SDK version 3.7[8] — One of the popular programming languages that makes use of an interpreter making it an interpreted, high-level language. This is used as the main developing language for D3driver because it has simpler syntax, easy usage, can be executed faster and works well for prototyping software.

---

[7]https://ngrok.com/docs
[8]https://www.python.org/
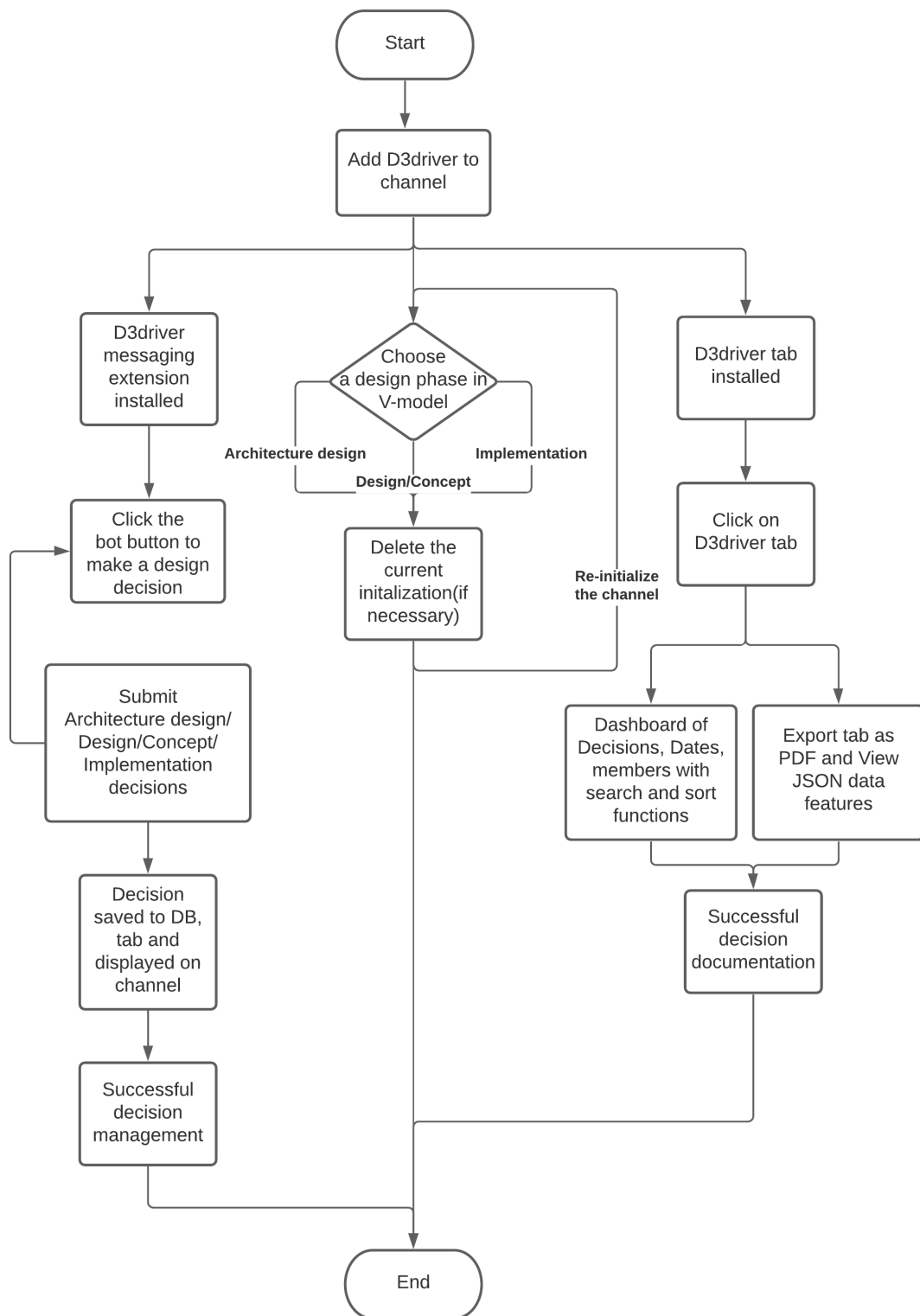
*Figure 5-4:     Workflow of D3driver*

- TinyDB[9] — As discussed in the architecture plan, this database will be used as decision database for D3driver. The design of the database is that it will have two tables to store channel related information and decision related information.

- HTML[10] & JavaScript[11] — D3driver tab uses these two technologies to render the card summary and other details. HTML to display the card information and JavaScript to make the page interactive.

- Visual studio code[12] — This is the best code editor created by Microsoft. Excellent features for code debugging, code review integrated with version control system(github). The biggest advantage is the availability of "Microsoft UI toolkit" extension with visual studio code specially used for bot development and management. It has some internal configurations with Azure that is done automatically for users and makes the process easier and faster.

This design is supposed to be the solution to poor or no documentation in a multi-disciplinary team. The design of D3driver aims at motivating team members to distinguish the most important design decisions from the typical conversations that they would have among their team. Designed in the best possible way to balance the simplicity, usability and functionality of the bot. The design also exhibits flexibility to choose the type of V-model process standard that the team wants to document. The design in ease of using the card with just one click is also the highlight and finally a dedicated tab to persist all the decisions together shows that the design is meant to decrease any effort of going through the long threads of decisions in a channel.

---

[9]https://tinydb.readthedocs.io/en/latest/

[10]https://www.w3.org/TR/html52/

[11]https://devdocs.io/javascript/

[12]https://code.visualstudio.com/

# 6        Implementation details

This chapter speaks about the implementation specifics of D3driver. This proof of concept demonstrates the possibility of execution of the solution idea that has been proposed in this thesis. All the source codes are available here[1]

## 6.1        Overview of D3driver

D3driver is a productivity bot that is uploaded as a custom app in Microsoft Teams tool. It is aimed at providing a feasible solution to teams that involve multiple domains while designing a product. It aids documenting design related decisions during a conversation between members from different domains. The team members discuss questions, answers, ideas, concepts, solutions, alternatives and decisions in their group chats. Since, decisions play a crucial role in product development, D3driver's features aid team members to track specifically design decisions document all the design decisions for later data retrieval.

The installation guidelines of D3driver is first explained. Later, the usage is also briefly re-explained to get more familiar with the bot. Finally, the technicalities of each feature and functionality is explained along with their code fragments for better and clear understanding of the implementation of D3driver. The source code for this implementation is based on the boilerplate code available from the official Microsoft's online guide called Bot builder samples[2]

## 6.2        D3driver Installation

The app package after uploading in MS Teams, D3driver's scope is within channels. The option "Add to team" can be selected to install the bot in the required channel. Once the D3driver is installed in a channel it sets up a messaging extension under the conversation area and a D3driver tab in the tab section as shown in the Figure 6-1.

---

[1]`https://github.com/bhargavimohan/Masterthesis_bot2021`
[2]`https://github.com/microsoft/BotBuilder-Samples`
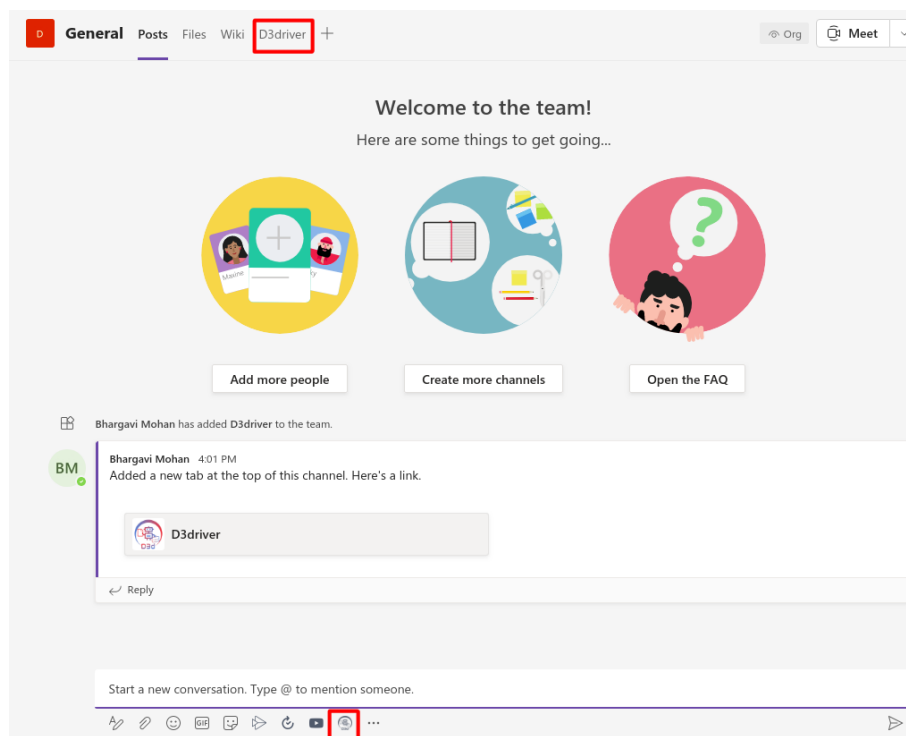
*Figure 6-1:      Message extension icon and D3driver tab*

### 6.2.1        Usage

- Firstly, a channel should be initialized. This means a team should first select one of the three phases in V-model for accessing design decision cards. The three phases will have 3 different decision cards hence a decision card in a channel depends upon the type of initialization. The command to initialize the channel is made available in the command list. The user has to invoke the bot by typing "@D3driver" in the compose message area and the list of commands will appear. The command `init` is used to initialize the channel

- Once the `init` command is used to invoke the bot, D3driver replies to the channel with an initial card. A user is required to select the appropriate design phase suitable to a particular channel.

- Depending on the choice, the decision cards can be accessed either by invoking the bot followed by the command — `Create card` or by hitting the messaging extension. The easiest and fastest way is to just hit the messaging extension.

- The different decision cards for architectural phase, design/conceptual phase and implementation phase are shown in Figures 6-8, 6-9 and 6-10 respectively.

- When users decide to decommission a channel, another command in the list `del` is available. By doing so, all the decisions discussed will be erased from the database. The users can choose to save the data to their local system before using `del` command by using the button "Export as PDF". It enables the members to save the entire D3driver tab as a PDF file. The filename will be in <channel name>_<vphase name> format.

## 6.3      D3driver and its components

Firstly, when a user sends any message by invoking the bot, the control goes to the main python file `app.py` as shown in Listing 6.1. It has a method called `messages()` that keeps snooping on any incoming messages and gives control to the object `BOT` of the class `TeamsMessagingExtensionsActionPreviewBot`.

*Listing 6.1:    app.py*

```python
BOT = TeamsMessagingExtensionsActionPreviewBot()
async def messages(req: Request) -> Response:
    # Main bot message handler.

    if "application/json" in req.headers["Content-Type"]:
        body = await req.json()
    else:
        return Response(status=HTTPStatus.UNSUPPORTED_MEDIA_TYPE)

    activity = Activity().deserialize(body)

    auth_header = req.headers["Authorization"] if "Authorization" in
        req.headers else ""

    invoke_response = await ADAPTER.process_activity(


        activity, auth_header, BOT.on_turn
    )
```

```
if invoke_response:
    return json_response(
        data=invoke_response.body, status=invoke_response.status
    )


return Response(status=HTTPStatus.OK)
```

This way, on any incoming message the control is transferred to the class. There are four important methods inside the class. 1) A method that takes control when the incoming message `init` 2) A method that takes control when the incoming message is `del` 3) A method that takes control when the messaging extension is invoked to access a decision card 4) A method that takes control when the submit button of the decision card is hit. This means that there is one method for each of the three commands and one method for submitting the decision card(this method is explained in the end, after decision cards are also discussed). These methods are explained in detail along with the details of the commands in the next section .

### 6.3.1        Commands

There are 3 useful commands that come handy while using this bot. `init` , `del` and `Create card` are shown in Figure 6-2. The result of typing "@D3driver" followed by either of the commands is discussed one-by-one in Sections 6.3.1.1 , 6.3.1.2, 6.3.1.3.

#### 6.3.1.1        init

When a channel needs to be initialized, the control jumps to the method `on_message_activity()` of class `TeamsMessagingExtensionsActionPreviewBot`. The following code snippet in Listing 6.2 explains how this request is handled. The variable `text_command` checks if the `D3driverinitcommand` is assigned the value `init`, if all the conditions meet, then the V-model phase selection card is posted in the channel. The card offers the channel members to choose the type of initialization that is required for their channel as shown in the Figure 6-3.

*Listing 6.2:      init command handling*

```
elif text_command == D3driverinitcommand:
                #vphase = 'Model-based'
```

```
result = database.find_channel_exists(channel_id)
if(len(result) == 0):
    card = create_vphase_card_editor()
    task_info = TaskModuleTaskInfo(
    card=card, height=450, title="Design decision card",
        width=500
        )
    message = MessageFactory.attachment(card)
    response_id = await turn_context.send_activity(message)
```
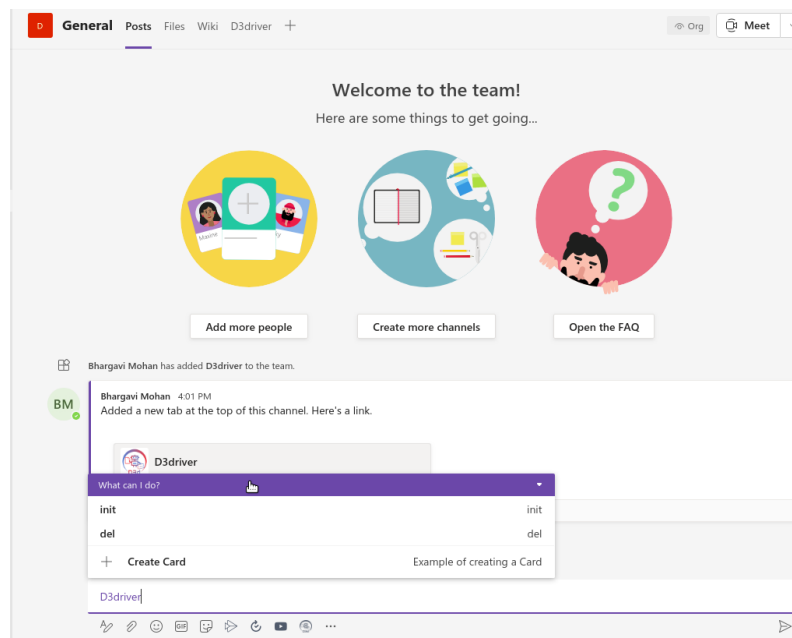


*Figure 6-2:    Commands*

The Listing 6.3 shows how the V-model phase card is created. It defined the `action` section and indicates that an action has to be taken when the choices are submitted. The body section defined some of the text formatting definitions and the `choices` that lists the available choices that are part the card.

*Listing 6.3:    V-model phase card*

```
def create_vphase_card_editor(
    user_text: str = None,
    is_multi_select: bool = False,
    option1: str = None,
```

*Figure 6-3:      V-model phase selection card*

```python
    option2: str = None,
    option3: str = None,
) -> Attachment:
    return CardFactory.adaptive_card(
        {
            "actions": [
                {
                    "data": {"submitLocation":
                        "messagingExtensionFetchTask", "type" :
                        "vmodelinit"},
                    "title": "Submit",
                    "type": "Action.Submit",
                }
            ],

            "body": [
                {
                "text": "Which phase in the V-model would you like to
                    document decisions?",
                "type": "TextBlock",
                "weight": "bolder",
                "wrap": True,
            },
                {
                    "choices": [
                        {
                            "title": "Architecture Design",
```

```
                                    "value": "Architecture Design"
                        },
                        {
                            "title": "Design/concept",
                            "value": "Design/concept"
                        },
                        {
                            "title": "Implementation",
                            "value": "Implementation"
                        }
                    ],
        "id": "Choices",
                        "isMultiSelect": is_multi_select,
                        "style": "expanded",
                        "type": "Input.ChoiceSet",
                    },
                ],
            "type": "AdaptiveCard",
             "version": "1.0",
            }
    )
```

### 6.3.1.2     **del**

This command is used in two scenarios. One is when the users decide that they no more need D3driver in their channel, it can be uninstalled by deleting the data present. An other likely scenario is when the users want to re-initialize the channel because they decide to use a different V-model phase for their project. If the team members would want to discuss a different set of design decisions, then users have to use the command del to reset their type of initialization. The control of the program is given to a different block in the method on_message_activity() as shown in Listing 6.4. The variable text_command checks if the D3driverdelcommand is assigned the value del. If the conditions are met, then the records of this channel and decisions are all deleted from the database. A message is posted on the channel informing the successful deletion.

*Listing 6.4:     del command handling*

```
elif text_command == D3driverdelcommand:
            database.delete_channel(channel_id)
```

```
database.delete_decision(channel_id)
reply = MessageFactory.text(
"This channel is no more initialized. All the design
    decsions(if discussed) has been deleted from the
    database. Plase click on 'Export to PDF' button in
    D3driver tab to save all the data to your local device.
    To re-initialize the channel, please enter "
"the init command"
)
await turn_context.send_activity(reply)
```

### 6.3.1.3        **Create card command / click on Messaging extension**

The `Create card` command triggers the messaging extension that is configured for D3driver which means choosing the `Create card` command and clicking on the D3driver's messaging extension has the exact same effect. The control is transferred to the same method in the code and D3driver reacts to both the events in the same way. There needs to be a little background knowledge on how it is possible to incorporate messaging extension service in the D3driver.

The Bot Framework SDK provides developers all the necessary tools, **adaptive cards**, messaging schema and communication protocols. To be able to use the SDK, a developer has to register the bot in the bot framework. The developer needs to provide some information during the registration like a name for the bot, optional endpoint etc. A Bot ID and password is generated at the end of the registration which will be required later. The bot registration of D3driver is shown in the Figure 6-4 and this is how the messaging extension is set up for D3driver.

In general, an action based messaging extension provides the team members with a popup. This popup can either gather information from the members or display a piece of information to the members. The gathered information is then inserted into the channel. The purpose of messaging extension in D3driver is to access decision cards and gather design decisions to be documented when any user inputs the decisions in the intended text-box area. Likewise, all the members of the channel can access decision cards to exchange design decision information and this is enabled by having the messaging extension configured. After the user hits submit button, the web service hosted by the bot framework is returned by posting the decision card to the channel.
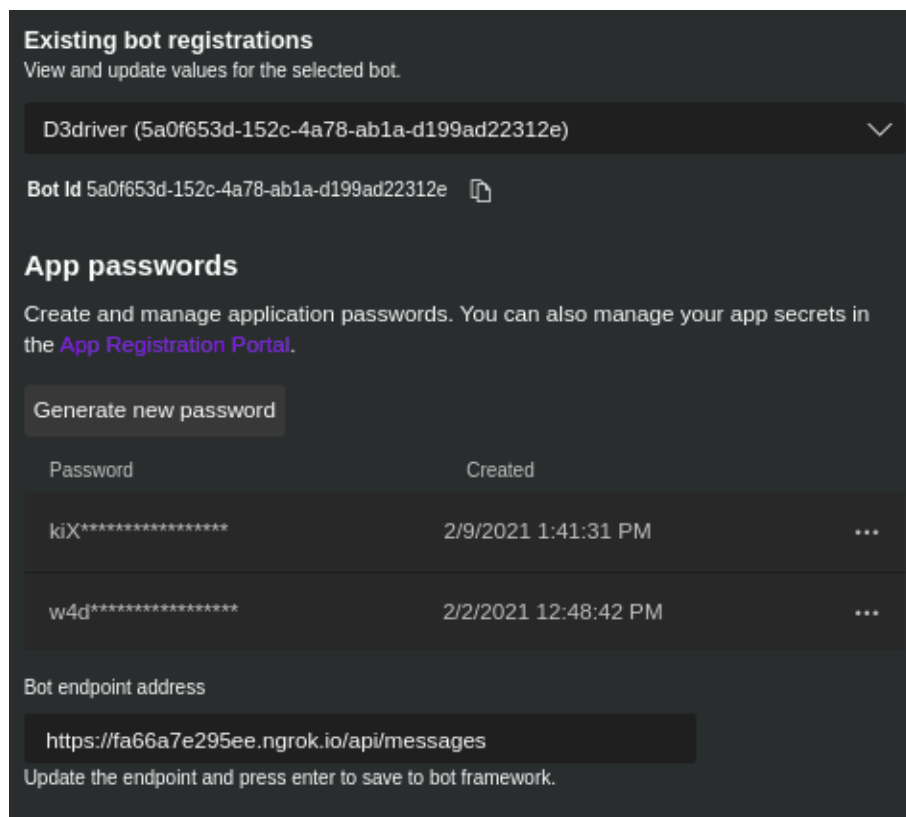
*Figure 6-4:      Bot registration*

Observe the Listing 6.5. The control of the program is now transferred to a different method i,e. to on_teams_messaging_extension_fetch_task(). This method examines which type of initialization a particular channel has and sends appropriate cards accordingly. The variable result is checked if it holds the value "Architecture Design" or "Design/Concept" or "Implementation" and three different cards namely "create_ad_adaptive_card_editor()" or "create_dc_adaptive_card_editor()" or "create_imp_adaptive_card_editor()" are posted to the channel depending on the value of result. The method also performs certain database operations to check if the channel is already initialized, if yes, relevant message is posted on the channel else requested card will be rendered.

*Listing 6.5:      messaging_extension_action_preview_bot.py*

```python
async def on_teams_messaging_extension_fetch_task(
      self, turn_context: TurnContext, action: MessagingExtensionAction
   ) -> MessagingExtensionActionResponse:
      if turn_context.activity.conversation.conversation_type !=
          'personal':
```

```python
channel_id =
    turn_context.activity.channel_data['channel']['id']
conversation_id = turn_context.activity.conversation.id #to
    distinguish 'Generals' of different teams
channel_name = turn_context.activity.conversation.name
if channel_name is None:
    channel_name = 'General'
result = database.find_channel_exists(channel_id)
if(len(result) == 1):
    if(result[0]["channel"]["vphase"] == 'Architecture
        Design'):
        card = create_ad_adaptive_card_editor()
        task_info = TaskModuleTaskInfo(
            card=card, height=450, title="Design decision
                card", width=500
        )
        continue_response =
            TaskModuleContinueResponse(value=task_info)
        return
            MessagingExtensionActionResponse(task=continue_response)
    elif(result[0]["channel"]["vphase"] == 'Design/concept'):
        card = create_dc_adaptive_card_editor()
        task_info = TaskModuleTaskInfo(
            card=card, height=450, title="Design decision
                card", width=500
        )
        continue_response =
            TaskModuleContinueResponse(value=task_info)
        return
            MessagingExtensionActionResponse(task=continue_response)
    elif(result[0]["channel"]["vphase"] == 'Implementation'):
        card = create_imp_adaptive_card_editor()
        task_info = TaskModuleTaskInfo(
            card=card, height=450, title="Design decision
                card", width=500
        )
        continue_response =
            TaskModuleContinueResponse(value=task_info)
```

```
                    return
                        MessagingExtensionActionResponse(task=continue_response)
            else:
                reply = MessageFactory.text(
                "Hi there! Please initialize your channel before accessing
                    the cards by using the init command."
                )
                await turn_context.send_activity(reply)
        else:
            reply = MessageFactory.text(
            "***D3driver*** is pleased to help you with decision cards in
                your required channels. Please access the bot using the
                default commands in any channel. ***Happy Documenting!***"
            )
            await turn_context.send_activity(reply)
```

Some of the important inbuilt classes and their object instances of the Bot framework
SDK that is operating in Listing 6.5 is explained in the following sub sections for deeper
understanding.

#### 6.3.1.3.1    Turncontext

By passing the class `TurnContext` as argument in the function calls as shown in 6.5,
it means new instance of the TurnContext class can be created. An object of the class
`TurnContext` is imported from the package `botbuilder-core` in the bot framework. This
object is helpful to get the specific context during the bot's turn for processing inputs or
triggering events. The context enables the developers to access all the necessary metadata
and values to handle activities that are upcoming. Furthermore, another class `BotAdapter`
handles the context object creation and other internal operations like authentication, sending
and receiving activities from Bot service. The `TurnContext` has a couple of constructors
and various properties and methods to perform specific actions.

### 6.3.1.3.2   MessagingExtensionAction

This class specifies the action brought by the messaging extension. It extends a class `TaskModulerequest`. This class calls upon the request payload. `MessagingExtensionAction` offers various parameters to access the user input data, user context, command ID, command context, message payload etc. It also has properties to get or set bot activity preview, get or set the ID of the command assigned by bot and gets or set message content sent as part of the command request etc.

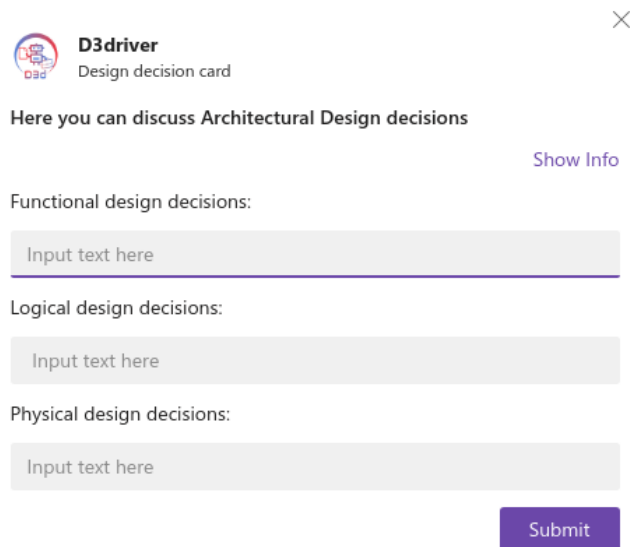### 6.3.1.3.3   MessagingExtensionActionResponse

This class returns a response to the messaging extension action. As seen in Listing 6.5 the class accepts a parameter `task` that returns the JSON for the Adaptive card to be rendered in the task module.

This marks the end of available commands in D3driver and how the corresponding function methods are called depending on the type of commands. Next, the three types of decision cards depending on the three types of initialization are discussed in the following section.

### 6.3.2        Decision cards

The decision cards in D3driver depends on the type of initialization. If the channel is initialized with "Architectural design" then the card shown in Figure 6-5 is popped up when `Create card` command is used or when messaging extension is clicked.

The method `create_ad_adaptive_card_editor` seen in Listing 6.6 is called when `Create card` command is used or when messaging extension is clicked. It returns an attachment i.e. the adaptive card of type architectural design. The attachment will contain the card with a suitable "contentType". A TypeError is caused if the card arguments are not python dicts. The adaptive cards are written in JSON format as seen in Listings 6.6, 6.7 and 6.8. The JSON has different parameters such as the field `actions` to define what has to be done when the submit button is clicked, the field `body` that defines the heading of the card, content of the sub headings and text-box areas and alignement, the field `Container` in the body indicates the meaning of each type of decision. The `Container` element introduces a toggle button called "Show info" and "Hide info" in UI of the

*Figure 6-5:    Architectural design decision card*

decision card as seen in Figure 6-5. The values of the fields deliver the styling and other formatting features to appear in the adaptive cards to make them look elegant and flawless.

*Listing 6.6:    Architectural design card*

```python
def create_ad_adaptive_card_editor(
    user_text1: str = None,
    user_text2: str = None,
    user_text3: str = None,
) -> Attachment:
    return CardFactory.adaptive_card(
        {
            "actions": [
                {
                    "data": {"submitLocation":
                        "messagingExtensionFetchTask"},
                    "title": "Submit",
                    "type": "Action.Submit",
                }
            ],
            "body": [
```

```
                {
                    "text": "Here you can discuss Architectural Design
                        decisions",
                    "type": "TextBlock",
                    "weight": "bolder",
                },
                {
            "type": "ColumnSet",
            "columns": [
                {
                    "type": "Column",
                    "selectAction": {
                        "type": "Action.ToggleVisibility",
                        "targetElements": [
                            "cardContent4",
                            "showInfo",
                            "hideInfo"
                        ]
                    },
                    "verticalContentAlignment": "Center",
                    "items": [
                        {
                            "type": "TextBlock",
                            "id": "showInfo",
                            "horizontalAlignment": "Right",
                            "color": "Accent",
                            "text": "Show Info",
                            "wrap": True
                        },
                        {
                            "type": "TextBlock",
                            "id": "hideInfo",
                            "horizontalAlignment": "Right",
                            "color": "Accent",
                            "text": "Hide Info",
                            "wrap": True,
                            "isVisible": False
                        }
                    ],
```

```
                "width": 1
            }
        ]
    },
    {
        "type": "Container",
        "id": "cardContent4",
        "isVisible": False,
        "items": [
            {
                "type": "Container",
                "items": [
                    {
                        "type": "TextBlock",
                        "text": "* ***Functional design decisions:***
                            Decisions related to functions describe
                            functionality of the system under
                            consideration independent of the solution.",
                        "isSubtle": True,
                        "wrap": True
                    },
                    {
                        "type": "TextBlock",
                        "text": "* ***Logical design decisions:***
                            Decisions related to logical architecture
                            describes a possible implementation of the
                            functions with defined solution principles.",
                        "isSubtle": True,
                        "wrap": True
                    },
                    {
                        "type": "TextBlock",
                        "text": "* ***Physical design decisions:***
                            Decisions related to the physical element
                            describes the actual implementation, e.g.
                            technical elements with part numbers,
                            concrete source code, etc.",
                        "isSubtle": True,
                        "wrap": True
```

```
                }
            ]
        }
    ]
},
        {"type": "TextBlock", "text": "Functional design
            decisions:"},
        {
            "id": "Question1",
            "placeholder": "Input text here",
            "type": "Input.Text",
            "value": user_text1,
            "isRequired": True,
            "errorMessage": "Any one feild is required"
        },

        {"type": "TextBlock", "text": "Logical design decisions:"},
        {
            "id": "Question2",
            "placeholder": " Input text here",
            "type": "Input.Text",
            "value": user_text2,
        },
        {"type": "TextBlock", "text": "Physical design
            decisions:"},
        {
            "id": "Question3",
            "placeholder": "Input text here",
            "type": "Input.Text",
            "value": user_text3,
        },
    ],
    "type": "AdaptiveCard",
    "version": "1.0",
}
)
```

If the channel is initialized with "Design/concept", then the card shown in Figure 6-6 shows up and the card is developed as shown in Listing 6.7. Note the descriptions of what type of decisions can be discussed using this decision template to guide the users.

*Listing 6.7:     Design/conceptual card card*

```python
def create_dc_adaptive_card_editor(
    user_text1: str = None,
    user_text2: str = None,
    user_text3: str = None,
) -> Attachment:
    return CardFactory.adaptive_card(
        {
            "actions": [
                {
                    "data": {"submitLocation":
                        "messagingExtensionFetchTask"},
                    "title": "Submit",
                    "type": "Action.Submit",
                }
            ],
            "body": [
                {
                    "text": "Here you can discuss Design/conceptual
                        decisions",
                    "type": "TextBlock",
                    "weight": "bolder",
                },
                {
            "type": "ColumnSet",
            "columns": [
                {
                    "type": "Column",
                    "selectAction": {
                        "type": "Action.ToggleVisibility",
                        "targetElements": [
                            "cardContent4",
                            "showInfo",
                            "hideInfo"
                        ]
```

```
                },
                "verticalContentAlignment": "Center",
                "items": [
                    {
                        "type": "TextBlock",
                        "id": "showInfo",
                        "horizontalAlignment": "Right",
                        "color": "Accent",
                        "text": "Show Info",
                        "wrap": True
                    },
                    {
                        "type": "TextBlock",
                        "id": "hideInfo",
                        "horizontalAlignment": "Right",
                        "color": "Accent",
                        "text": "Hide Info",
                        "wrap": True,
                        "isVisible": False
                    }
                ],
                "width": 1
            }
        ]
    },
    {
        "type": "Container",
        "id": "cardContent4",
        "isVisible": False,
        "items": [
            {
                "type": "Container",
                "items": [
                    {
                        "type": "TextBlock",
                        "text": "* ***Design definition decisions:***
                            Decisions related to technology
                            management,design objectives and design
                            definition strategy, including the need for
```

```
                                  and requirements of any enabling systems,
                                  products, or services.",
                          "isSubtle": True,
                          "wrap": True
                  },
                  {
                          "type": "TextBlock",
                          "text": "* ***Design characteristics & Enablers
                                  decisions:*** Decisions related to design
                                  characteristics for the architectural
                                  entities that ensures that the design
                                  characteristics are feasible. Design
                                  enablers such as models (physical and
                                  analytical), design heuristics, etc. are
                                  decided.",
                          "isSubtle": True,
                          "wrap": True
                  },
                  {
                          "type": "TextBlock",
                          "text": "* ***Design Alternatives decisions:***
                                  Decisions related to assessing options for
                                  the system element to be developed using
                                  selection criteria. The most appropriate
                                  alternatives decision is made to develop the
                                  system element.",
                          "isSubtle": True,
                          "wrap": True
                  }
              ]
          }
      ]
  },
      {"type": "TextBlock", "text": "Design definition
          decisions:"},
      {
          "id": "Question1",
          "placeholder": "Input text here",
          "type": "Input.Text",
```

```
                    "value": user_text1,
                },
                {"type": "TextBlock", "text": "Design characteristics &
                    Enablers decisions:"},
                {
                    "id": "Question2",
                    "placeholder": "Input text here",
                    "type": "Input.Text",
                    "value": user_text2,
                },
                {"type": "TextBlock", "text": "Design Alternatives
                    decisions:"},
                {
                    "id": "Question3",
                    "placeholder": "Input text here",
                    "type": "Input.Text",
                    "value": user_text3,
                },
            ],
            "type": "AdaptiveCard",
            "version": "1.0",
        }
    )
```

If the channel is initialized with "Implementation", then the card in Figure 6-7 appears to the users. The card is developed as shown in Listing 6.8. Please observe the meanings of each decision type described to guide the users.

*Listing 6.8:    Implementation card*

```python
def create_imp_adaptive_card_editor(
    user_text1: str = None,
    user_text2: str = None,
    user_text3: str = None,
) -> Attachment:
    return CardFactory.adaptive_card(
        {
            "actions": [
                {
```

*Figure 6-6:     Design/concept decision card*

```
        "data": {"submitLocation":
            "messagingExtensionFetchTask"},
        "title": "Submit",
        "type": "Action.Submit",
    }
],
"body": [
    {
        "text": "Here you can discuss Implementational design
            decisions",
        "type": "TextBlock",
        "weight": "bolder",
    },
    {
"type": "ColumnSet",
"columns": [
    {
```

*Figure 6-7:    Implementation decision card*

```
"type": "Column",
"selectAction": {
    "type": "Action.ToggleVisibility",
    "targetElements": [
        "cardContent4",
        "showInfo",
        "hideInfo"
    ]
},
"verticalContentAlignment": "Center",
"items": [
    {
        "type": "TextBlock",
        "id": "showInfo",
        "horizontalAlignment": "Right",
        "color": "Accent",
        "text": "Show Info",
        "wrap": True
    },
    {
```

```
                              "type": "TextBlock",
                              "id": "hideInfo",
                              "horizontalAlignment": "Right",
                              "color": "Accent",
                              "text": "Hide Info",
                              "wrap": True,
                              "isVisible": False
                        }
                    ],
                    "width": 1
                }
            ]
        },
        {
            "type": "Container",
            "id": "cardContent4",
            "isVisible": False,
            "items": [
                {
                    "type": "Container",
                    "items": [
                        {
                            "type": "TextBlock",
                            "text": "* ***Mechanical design decisions:***
                                Decisions related to mechanics like
                                displacement, velocity, acceleration, force
                                , torque etc.",
                            "isSubtle": True,
                            "wrap": True
                        },
                         {
                            "type": "TextBlock",
                            "text": "* ***Electrical design decisions:***
                                Decisions related to electronics like
                                electric variables, voltage, current,
                                electric field strength etc.",
                            "isSubtle": True,
                            "wrap": True
                        },
```

```
            {
                "type": "TextBlock",
                "text": "* ***Software design decisions:***
                    Decisions related to digital information
                    process like logical operations, algorithms,
                    database etc.",
                "isSubtle": True,
                "wrap": True
            }
        ]
    }
],
},
    {"type": "TextBlock", "text": "Mechanical design
        decisions:"},
    {
        "id": "Question1",
        "placeholder": "Input text here",
        "type": "Input.Text",
        "value": user_text1,
    },
    {"type": "TextBlock", "text": "Electrical design
        decisions:"},
    {
        "id": "Question2",
        "placeholder": "Input text here",
        "type": "Input.Text",
        "value": user_text2,
    },
    {"type": "TextBlock", "text": "Software design
        decisions:"},
    {
        "id": "Question3",
        "placeholder": "Input text here",
        "type": "Input.Text",
        "value": user_text3,
    },
],
"type": "AdaptiveCard",
```

```
        "version": "1.0",
    }
)
```

This ends the discussion about the three types of cards that are made available in D3driver to discuss and document design decisions. The following sub section deals with the action taken by D3driver when the "submit" button is clicked in each of the decision cards. The same action is taken in all three cases and is addressed as follows.

### 6.3.2.1     **Submit button**

When the users insert their discussion topic in the cards and hits the submit button, the control is transferred to a method that handles this event. The four important methods of the class `TeamsMessagingExtensionsActionPreviewBot` were listed earlier. This section describes the forth method `on_teams_messaging_extension_submit_action()` that performs two tasks as shown in Listing 6.9. It captures the decision type, decision maker's name, date and the design decision topic and makes a database store operation to save all the information of the user that is about to submit the card. Secondly, the card is posted on the channel. The card is called "Preview card" and this way the sender is making his decision public by clicking on submit button. The "preview card" can be visualized in the Figures 6-8,6-9 and 6-10. One card from each type of initialization is depicted. The method also handles a scenario when users do not type anything and hits submit, an error message is posted. The "preview card" is developed as shown in Listing 6.10.

*Listing 6.9:     Submit button handling*

```python
async def on_teams_messaging_extension_submit_action( # pylint:
   disable=unused-argument
       self, turn_context: TurnContext, action: MessagingExtensionAction
   ) -> MessagingExtensionActionResponse:

       channel_id = turn_context.activity.channel_data['channel']['id']
       year = str(turn_context.activity.local_timestamp.year)
       month = str(turn_context.activity.local_timestamp.month)
       day = str(turn_context.activity.local_timestamp.day)
       decisiondate = year + "-" + month + "-" + day
       channel_name = turn_context._activity.conversation.name
       if channel_name is None:
```

```python
        channel_name = 'General'
    result = database.find_channel_exists(channel_id)
    if(result[0]["channel"]["vphase"] == 'Architecture Design'):
        a="Functional design decision"
        b="Logical design decision"
        c="Physical design decision"
    elif(result[0]["channel"]["vphase"] == 'Design/concept'):
        a="Design definition desisions"
        b="Design characteristics and enablers decisions"
        c="Design alterantives decisions"
    elif(result[0]["channel"]["vphase"] == 'Implementation'):
        a="Mechanical design decisions"
        b="Electrical design decisions"
        c="Software design decisions"


    user_text1 = action.data["Question1"],
    user_text2 = action.data["Question2"],
    user_text3 = action.data["Question3"],

    if(user_text1[0] == ''):
        a = ''
    if(user_text2[0] == ''):
        b = ''
    if(user_text3[0] == ''):
        c = ''

    memberid = turn_context._activity.from_property.name
    card = create_adaptive_card_preview(
        user_text1[0],
        user_text2[0],
        user_text3[0],
        a,
        b,
        c,
        memberid,
    )

    # db entries
```

```python
if(result[0]["channel"]["vphase"] == 'Architecture Design'):
    database.insert_decision(channel_id,channel_name,memberid,decisiondate,
    a,user_text1,b,user_text2,c,user_text3)
elif(result[0]["channel"]["vphase"] == 'Design/concept'):
    database.insert_decision(channel_id,channel_name,memberid,decisiondate,
    a,user_text1,b,user_text2,c,user_text3)
elif(result[0]["channel"]["vphase"] == 'Implementation'):
    database.insert_decision(channel_id,channel_name,memberid,decisiondate,
    a,user_text1,b,user_text2,c,user_text3)


if (user_text1[0] or user_text2[0] or user_text3[0] != ''):
    message = MessageFactory.attachment(card)
    await turn_context.send_activity(message)

    return MessagingExtensionActionResponse()
else:
    reply = MessageFactory.text(
    "Error. No decision made"
    )
    await turn_context.send_activity(reply)
```
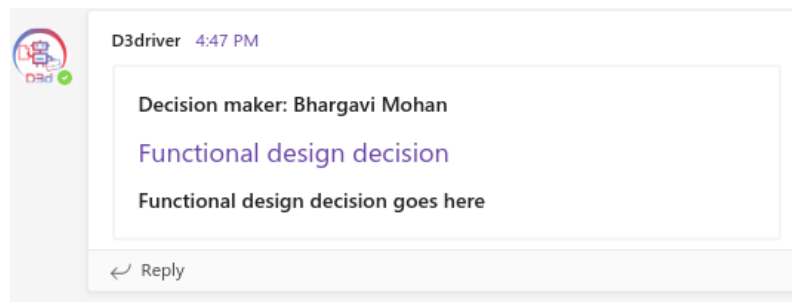


*Figure 6-8:    Functional design decision posted on channel*

*Listing 6.10:    Preview card*

```python
def create_adaptive_card_preview(
    user_text1: str = None,
    user_text2: str = None,
    user_text3: str = None,
    a: str = None,
    b: str = None,
```
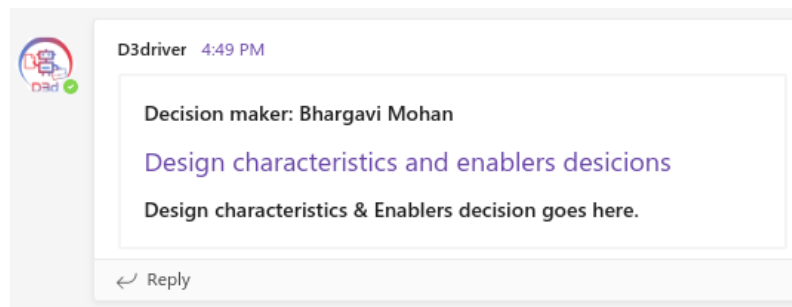
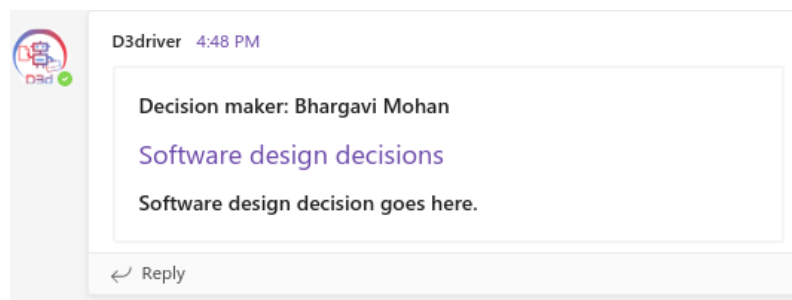*Figure 6-9:     Design characteristics and enablers decision posted on channel*



*Figure 6-10:     Software design decision posted on channel*

```python
    c: str = None,
    memberid: str = None,
) -> Attachment:
    return CardFactory.adaptive_card(
        {
            "body": [
                {
                    "text": "Decision maker: {}".format(memberid),
                    "type": "TextBlock",
                    "weight": "bolder",
                },
                {
                    "text": "{}".format(a),
                    "type": "TextBlock",
                    "id": "Question1",
                    "color": "Accent",
                    "size": "Large"
                },
                {
```

```
                "text": "{}".format(user_text1),
                "type": "TextBlock",
                "id": "Question1",
                "weight": "bolder"
            },


            {
                "text": "{}".format(b),
                "type": "TextBlock",
                "id": "Question2",
                "color": "Accent",
                "size": "Large"
            },
            {
                "text": "{}".format(user_text2),
                "type": "TextBlock",
                "id": "Question1",
                "weight": "bolder"
            },
            {
                "text": "{}".format(c),
                "type": "TextBlock",
                "id": "Question3",
                "color": "Accent",
                "size": "Large"
            },
            {
                "text": "{}".format(user_text3),
                "type": "TextBlock",
                "id": "Question1",
                "weight": "bolder"
            },

        ],
        "type": "AdaptiveCard",
        "version": "1.0",
    }
)
```

## 6.4         D3driver Tab

There are two types of tabs in Microsoft teams. 1) Static tabs and 2) Configurable tabs. Configurable tab is the focus in this thesis. It lets one to set the content of the tab dynamically each time the bot is added to a channel. It is nothing but a web-based HTML page generated inside a teams configurable tab. The content URL must be set properly to see a working configurable tab.(The endpoint URL can be submitted while the bot registration explained in 6.3.1.3)

D3driver installs a configurable tab upon bot installation and the welcome page is previewed before the tab is installed as shown in Figure 6-11. Once the tab is saved to a channel, all the channel specific data appears in the configured tab. The tab serves like a dashboard that displays all the design decisions discussed along with its other parameters in a team. Every channel will has its own D3driver tab and the design decisions, channel initialization type and other important information present in a tab will pertain to its respective channel. The tab will additionally have a search box, sort button and a button to export the tab as a PDF and a button to view the meta data in JSON format as shown in Figure 6-13.
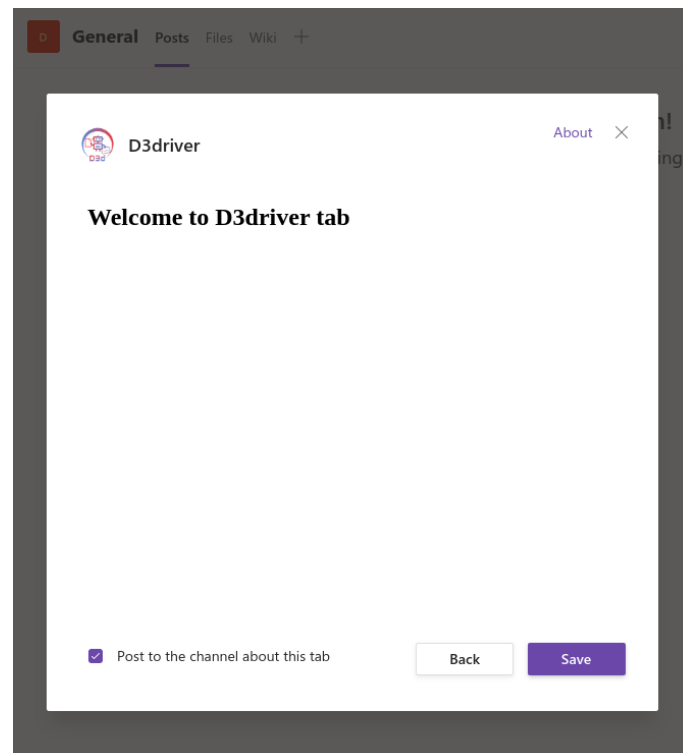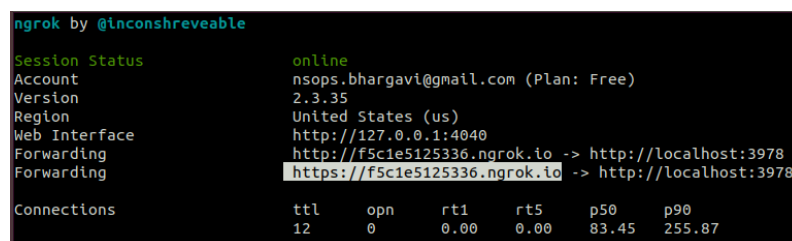


*Figure 6-11:     Welcome tab*

### 6.4.1        Ngrok

The way the content is published in the tab is by using an `ngrok` tunnel. Using `ngrok`, the local endpoint is exposed to the internet hence making the HTML content available publicly. The `ngrok` providing public URL in terminal can be seen in Figure 6-12. The command to expose the local end point is right below.

```
$ ngrok http -host-header=rewrite 3978
```



*Figure 6-12:     Ngrok endpoint*

### 6.4.2        Jinja

Furthermore, for the purpose of templating HTML, Jinja is used. It is a web template tool for python coding. This well known template engine is responsible for populating all the mark up, source code, database records in the D3driver tab. According to the Listing 6.11, a page is rendered by applying `@aiohttp_jinja2.template()` decorator to the web-handler. The method `dashboard` returns a jnja2 context. The decorator now presents the context as the web response. The python list variables `response_obj_tables` and `response_obj_decisions` capture the required data records from the database to pass it on the to the desired HTML file(dashboard.html).

*Listing 6.11:     Dashboard method*

```python
@aiohttp_jinja2.template("dashboard.html")
async def dashboard(request):
    try:
        param = request._rel_url.query_string
        param = param.strip('channelid=')
        response_obj_tables = database.get_channel_details(param)
        response_obj_decisions = database.get_all_decisions(param)
        ## rErroreturn a success json response with status code 200 i.e.
            'OK'
```
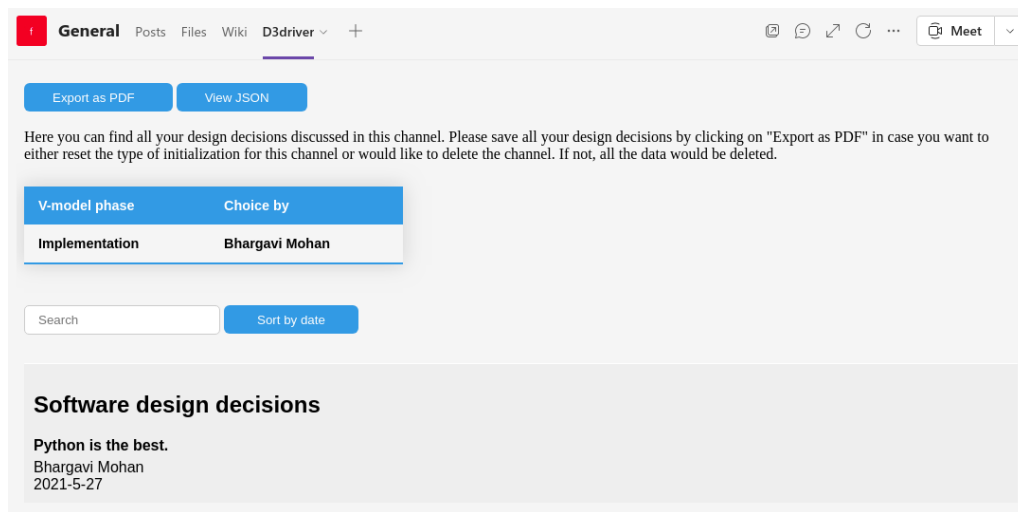
*Figure 6-13:     D3driver tab installed in a channel*

```python
    return {"decisions_list":
        response_obj_decisions,"tables_list":response_obj_tables}
except Exception as e:
    ## Bad path where
```

### 6.4.3        AIOHTTP

A small web server is created using `aiohttp` to support the functionality of viewing the meta data of any channel. The meta data is the raw data that is fetched from the database. The meta data is present in the JSON format. Upon click of "View JSON" in D3driver tab, a user is redirected to their default browser to view and further work with it to edit or copy to the local system. A request handler accepts `request` as its only argument and returns the JSON data in response as shown in Listing 6.12.

*Listing 6.12:     View JSON button*

```python
async def jsonreply(request):
    try:
        param = request._rel_url.query_string
        param = param.strip('channelid=')
        response_obj_tables = database.get_channel_details(param)
        response_obj_decisions = database.get_all_decisions(param)
        return web.json_response({"decisions_list":
            response_obj_decisions,"tables_list":response_obj_tables})
```

```
except Exception as e:
    return web.json_response({"error": e})
```

### 6.4.4      JavaScript

Two scripts have been used for the functionality of search box, sort button and export button. The library list.js has been used for search and date sort functions. For the export button, html2pdf javascript library has been used. A sample javascript is shown in Listing 6.13 to observe how the PDF download script works.

*Listing 6.13:    PDF download script*

```
<script>
  window.onload = function () {
    document.getElementById("download")
        .addEventListener("click", () => {
            const invoice = this.document.getElementById("invoice");
            console.log(invoice);
            console.log(window);
            var opt = {
                margin: 1,
                filename: "{{ tables_list.name | safe}}_{{
                    tables_list.vphase | safe}}",
                image: { type: 'jpeg', quality: 0.98 },
                html2canvas: { scale: 2 },
                jsPDF: { unit: 'in', format: 'letter', orientation:
                    'portrait' }
            };
            html2pdf().from(invoice).set(opt).save();
        })
}

</script>
```

### 6.5      Database structure

The storage for D3driver implementation is document oriented database. The data is stored as python dictionary in TinyDB. TinyDB makes use of the Python JSON module hence

query to TinyDB fetches JSON data in return. TinyDB extends its support in creation of DB tables. The data modeling in tables is as same shown in Listing 6.14. Every record in the dictionary uses the name of the table as **key** and documents dictionary as **value**. Therefore, every record is stored as a **key-value** pair. The dictionary of documents contain document IDs as keys and the documents as values.

*Listing 6.14:     Data in TinyDB tables*

```
{
    'table1': {
        0: {document...},
        1: {document...},
    },
    'table2': {
        ...
    }
}...
```

For D3driver's implementation, two such tables are used.

1. `channelstable`— This table stores the list of all the initialized channels. It stores other important data as shown in Listing 6.15. It represents one entry in the table. The `channelid` is a unique string of characters that is assigned to each channel. The `name` is the name of the channel. `vphase` is the type of initialization of that channel. `memebername` is the member who initialized the channel to `vphase` value. `date` represents the day on which the channel was initialized.

*Listing 6.15:     Initialized channels data*

```
"Initializedchannelsanddata": {
    "1": {
        "channel": {
            "channelid":
                "19:5642f28c1510444d9af14304ce7ce158@thread.tacv2",
            "name": "bottest",
            "vphase": "Design/concept",
            "membername": "Bhargavi Mohan",
            "date": "2021-3-10"
        }
    }
```

2. `decisionstable`— This table stores all the design decisions of every initialized channel. The Listing 6.16 shows a single record of this table. The `channelid` is a unique string of characters that is assigned to each channel. The `name` is the name of the channel. `membername` represents the decision maker's name. `date` represent the day on which a particular decision has been made. `decisionname_a`, `decisionname_b` and `decisionname_c` corresponds the name of design decisions. This depends once again on the type of initialization of the channel. Since the channel's name is "bottest" and it's type of initialization is "**Design/concept**" (from vphase values in Listing 6.15), thus the `decisionname_a`, `decisionname_b` and `decisionname_c` would be **Design definition decisions**, **Design characteristics & Enablers decisions** and **Design Alternatives decisions** respectively. In the example shown in Listing 6.16, only decision pertaining to **Design characteristics and enablers decisions** has been made, hence there is an entry against `decisionname_b` The records with the key names `decision_a`, `decision_b` and `decision_c` will be paired with their respective values which is the actual decision topic. In the below example, `decision_b` has some value tagged to it.

*Listing 6.16:    Design decisions table*

```
"Designdecisions": {
    "1": {
        "decisions": {
            "channelid":
                "19:5642f28c1510444d9af14304ce7ce158@thread.tacv2",
            "name": "bottest",
            "membername": "Bhargavi Mohan",
            "date": "2021-3-10",
            "decisionname_a": "",
            "decision_a": "",
            "decisionname_b": "Design characteristics and
                enablers decisions",
            "decision_b": "This is a place where Design
                characteristics and enablers decisions will be
                discussed",
            "decisionname_c": "",
            "decision_c": ""
        }
    }
```

## 6.6        Sequential workflow

The sequence diagrams are a well known way of demonstrating the interactions and operations in a process. It also depicts the order of interactions or messages sent and received.

The 3 sequence diagrams can be observed in Figure 6-14, 6-15 and 6-16. These basically represent the sequence of actions,reactions and exchange of messages between different functions. The figures show the sequential workflow of what happens when users invoke the bot by using the three basic commands of D3driver. The content in the rectangular boxes depicts the file names and the messages on the arrow depicts what is the action taking place. The vertical lines are called lifelines and the arrows represent asynchronous messages and replies.
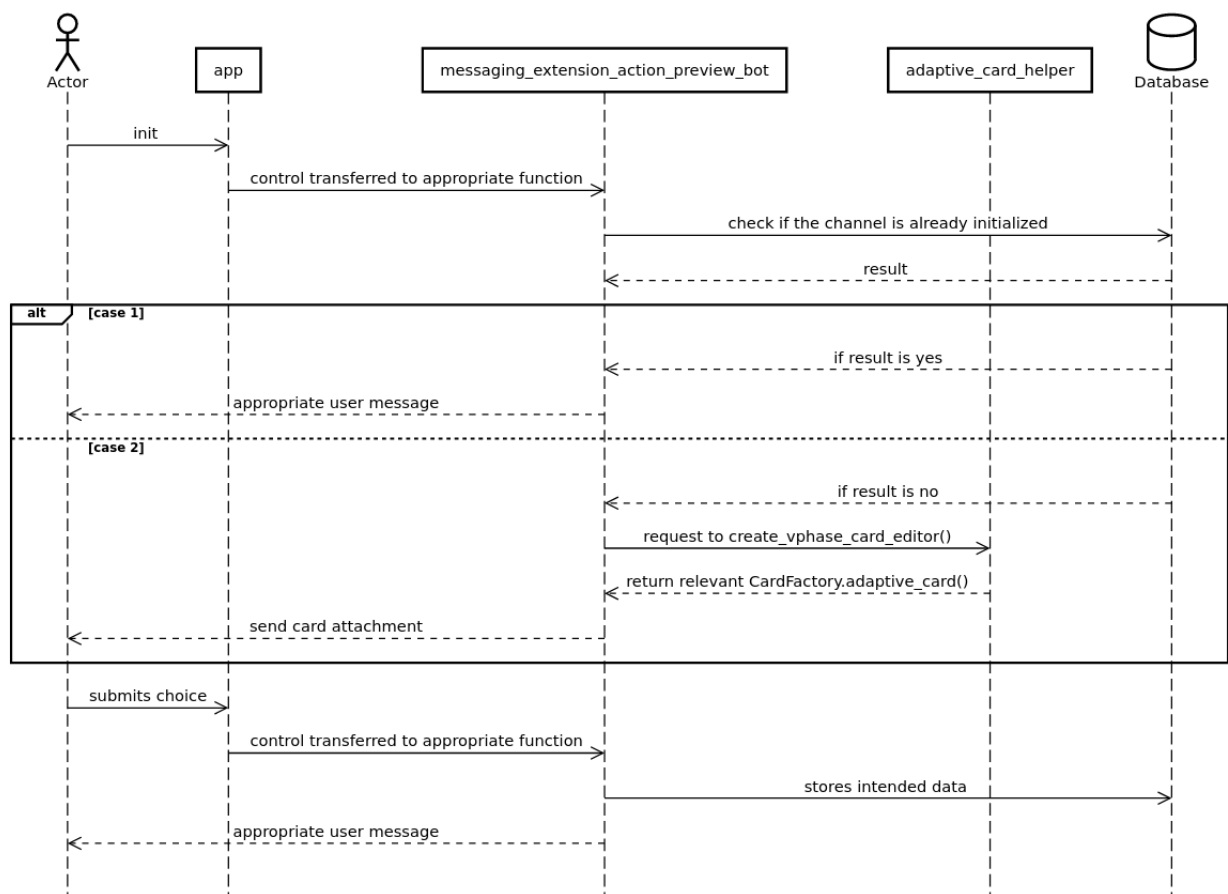


*Figure 6-14:     @D3driver init*

In Figure 6-14, the rectangular big box named "alt" represents that there is a condition being checked. In Figure 6-15, the big rectangular box represents that the event can happen in a loop until the user wishes to.
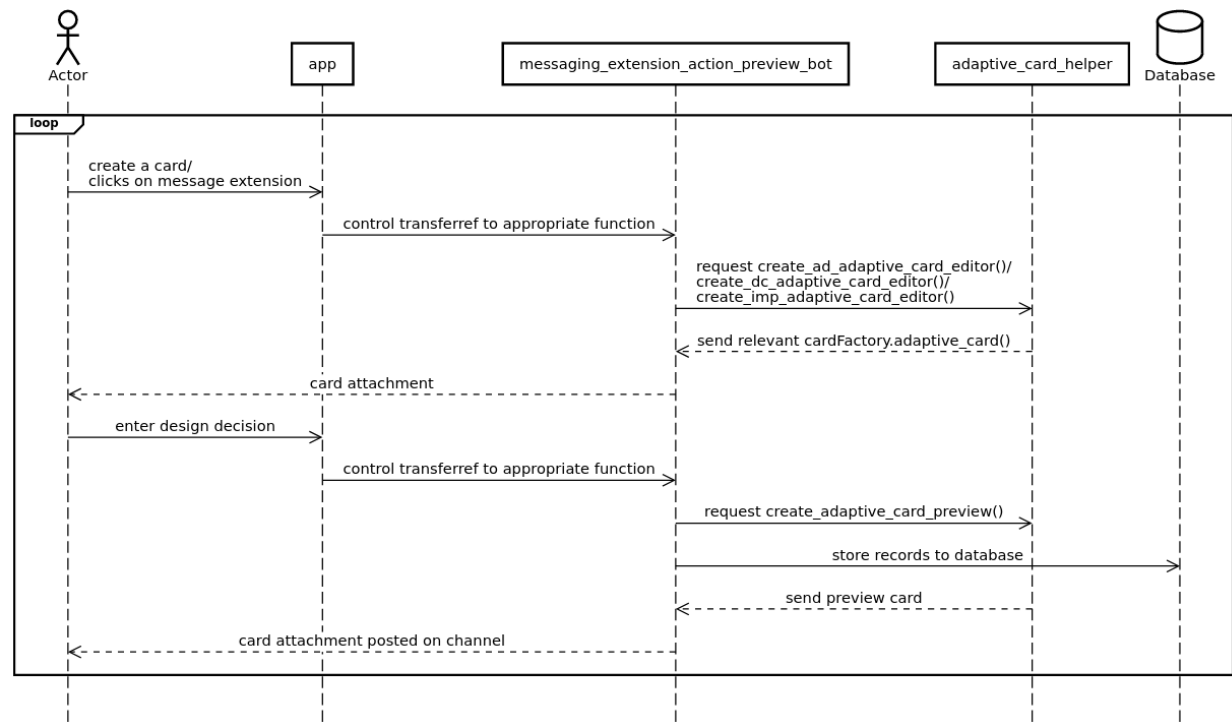


*Figure 6-15:    Create card*

## 6.7        Manifest schema

All the apps available in MS teams app store has to define an app manifest file. This file is a compressed file of one JSON file named as `manifest.JSON`, and two PNG versions of colored bot icon and an outlined bot icon of 192*192 pixels. In order to be able to upload a custom bot, the developer should prepare a zipped package of all the three. ***Appstudio***[3] is already available in the MS teams app store, using which, one can easily create and edit the application manifest.

---

[3]`https://microsoft.github.io/botframework-solutions/clients-and-channels/`
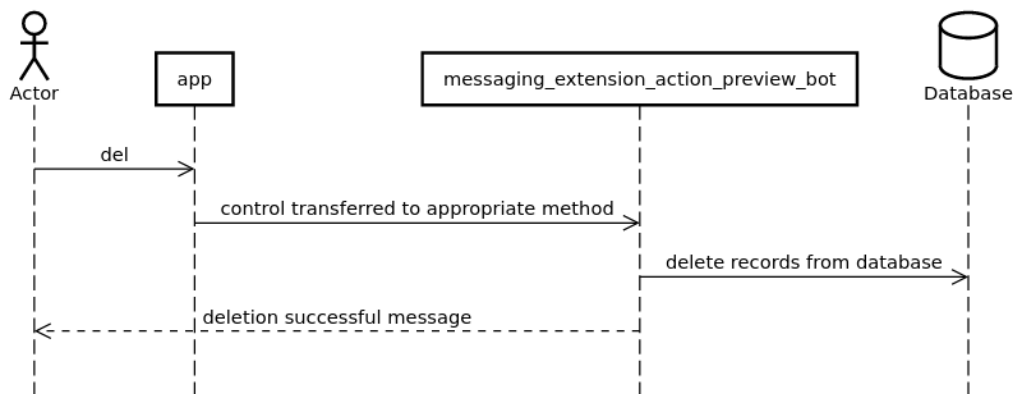  `tutorials/enable-teams/4-create-app-manifest/`

*Figure 6-16:    @D3driver del*

The manifest JSON file is the description of how the bot should be incorporated into Microsoft teams tool. This JSON should abide by the standard schema[4] define by Microsoft. The file provides various options for attributes in its JSON body. The attributes in the JSON leads to its applicable sources such as Microsoft BOT ID, BOT password(that is generated during bot registration), a configuration URL to any static/configurable tabs. It is in this file the developer mentions if the bot should install a messaging extension , have configurable tabs , have default commands etc. There are other mandatory fields in the JSON body that should have valid values. Only when all the required fields are given the right values, the custom app(bot) can be uploaded and can be used in Microsoft teams. The app manifest of D3driver can be observed in Listing 6.17. Observe the D3driver bot ID, bot full description, configuration URL that renders the D3driver tab, messaging extension configured and the scopes in which the bot is allowed to be added.

*Listing 6.17:    Manifest file*

```
{
  "$schema":
      "https://developer.microsoft.com/en-us/json-schemas/teams/v1.8/
  MicrosoftTeams.schema.json",
  "manifestVersion": "1.8",
  "version": "1.0",
  "id": "5a0f653d-152c-4a78-ab1a-d199ad22312e",
  "packageName": "com.microsoft.teams.samples",
  "developer": {
    "name": "UPB - Bhargavi Mohan",
```

---

[4]https://developer.microsoft.com/en-us/json-schemas/teams/v1.8/MicrosoftTeams.
  schema.json

```
      "websiteUrl": "https://dev.botframework.com",
      "privacyUrl": "https://privacy.microsoft.com",
      "termsOfUseUrl":
          "https://www.microsoft.com/en-us/legal/intellectualproperty/copyright/
      default.aspx"
    },
    "icons": {
      "color": "icon-color.png",
      "outline": "icon-outline.png"
    },
    "name": {
      "short": "D3driver",
      "full": "Design Decision Document driver"
    },
    "description": {
      "short": "Bot that aids documenting design decisions",
      "full": "D3driver drives the design decision documentation in the
          easiest way possible. This bot is exclusively designed to be used
          in a Mechatronics team. The design phase is an integral step in
          systems engineering domain. In an inter-disciplinary team, the
          design stage during the lifecycle of a product is extended across
          different phases of V-model and hence the design decisions are
          made throughout architectural design phase , conceptual design
          phase and even during the implementation phase. All these design
          decisions and decision makers should be distinguished from
          regualar or unimportant discussions that happen within a team and
          put together for later quick access and that is exactly what
          D3driver assists the users with. D3driver installs a messaging
          extension and a tab. The messaging extension is to access decision
          cards where user can choose from which phase of the V-model does
          the user want to document the design decisions. The tab serves
          like a dashboard that displays all the design decisions discusssed
          in a team."
    },
    "accentColor": "#FFFFFF",
    "configurableTabs": [
      {
        "configurationUrl": "https://f5c1e5125336.ngrok.io/config",
        "canUpdateConfiguration": true,
```

```
    "scopes": [
      "team"
    ],
    "context": [
      "channelTab"
    ]
  }
],
"bots": [
  {
    "botId": "5a0f653d-152c-4a78-ab1a-d199ad22312e",
    "scopes": [
      "personal",
      "team"
    ],
    "commandLists": [
      {
        "scopes": [
          "team"
        ],
        "commands": [
          {
            "title": "init",
            "description": "init"
          },
          {
            "title": "del",
            "description": "del"
          }
        ]
      }
    ],
    "supportsFiles": false,
    "isNotificationOnly": false
  }
],
"composeExtensions": [
  {
    "botId": "5a0f653d-152c-4a78-ab1a-d199ad22312e",
```

```json
    "canUpdateConfiguration": false,
    "commands": [
     {
        "id": "createWithPreview",
        "type": "action",
        "title": "Create Card",
        "description": "Example of creating a Card",
        "initialRun": false,
        "fetchTask": true,
        "context": [
          "commandBox",
          "compose",
          "message"
        ],
        "parameters": [
          {
            "name": "param",
            "title": "param",
            "description": ""
          }
        ]
      }
    ]
  }
],
"permissions": [
  "identity",
  "messageTeamMembers"
],
"validDomains": [
"7b22dd51bc42.ngrok.io",
"*.cdnjs.cloudflare.com",
 "static2.sharepointonline.com",
 "secure.aadcdn.microsoftonline-p.com",
 "code.jquery.com",
 "statics.teams.microsoft.com",
 "*.microsoftonline.com",
 "ajax.googleapis.com",
 "*.bing.com",
```

```
  "*.google.com",
  "*.statics.teams.cdn.office.net"
 ]
}
```

## 6.8        Dockerization of D3driver

The process of wrapping a software application along with it's dependencies inside a virtual container is the basic idea behind dockerization. The main advantage of this is that it lets the developers to build light-weight, scalable and portable software containers that makes development, testing and deployment easier. Interested readers can refer [RBA17] for detailed information on dockers and their advantages.

Docker image contains all the required files like dependency files, source and important libraries packed together. To build a docker image of D3driver, a *Dockerfile* is created. The docker file contains the commands that will be executed when the *docker-build* command is executed in the terminal as shown in Listing 6.18.

*Listing 6.18:     Docker build command*

```
docker build -f Dockerfile -t python-d3d-docker-bot .
```

The D3driver docker image creates a docker container named *python-d3d-docker-bot* when the *docker-run* command is executed as shown in Listing 6.19. This container will have the entire D3driver application that can be run in any given environment irrespective of any operating system. Once the docker-run command is executed in the terminal, the container is started and D3driver starts running. With this, the bot can be accessed in Microsoft teams.

*Listing 6.19:     Docker run command*

```
docker run --rm -it -p 3978:3978 -v
   /home/bhargavi/Documents/Masterthesis_bot2021/src/database:"/app/database"
   python-d3d-docker-bot
```

The implementation of D3driver is now complete. The next chapter involves the validation and verification report of the implemented software.

# 7        Verification and Validation of D3driver

This document details the verification and validation of D3driver. In general, verification process is done to check if all the requirements has been incorporated correctly into the software product whereas validation process is to get the software product tested with the stakeholder/customers to confirm that the software product is according to their requirements. In this thesis, the two types of tests are done in the following way:

1. Verification process based on the *Evaluation Criteria* that was described to screen the existing bots in Section 4.3.

2. Validation process based on the conversation simulation between two members of the mechatronics domain.

## 7.1        Evaluation criteria — Revisited

The Evaluation criteria that was described (based on the Requirement characteristics) in Chapter 4 were used to check the suitability of existing bots in the context of employee motivation towards documenting design decisions. In this section, D3driver will go through the same set of criteria to get through the verification process as shown in Table 7-1.

| Evaluation criteria | D3driver |
|---|---|
| Workspace - MS teams | Operational in MS teams(prototype) |
| Message tagging/labeling(VDI specific) | Provides decision templates in 3 important V-model design phases |
| Design decision management | Clearly distinguishes normal conversations from design decisions |
| Design decision documentation | Offers a separate tab to view discussed design decisions |
| Design decision Knowledge management | Allows members to save a copy of the discussed decisions as PDF |
| Easy metadata retrieval | A workaround to retrieve raw data as JSON is enabled |

*Table 7-1:    D3driver against the evaluation criteria*

## 7.2        Conversation simulation using D3driver

The validation process was done through simulating an "Engineering Situation" between two stakeholders using D3driver. The end goal of this exercise was to mimic the conversation as if in a product development environment. This helped in establishing

helpful user stories to realize D3driver as an effective solution to the problem statement of this thesis.

The agenda of this workshop was as follows

- Initial setup — A channel was created in Microsoft teams for the validation purpose. The bot D3driver was added to the channel. The supervisors(members) were also added to the channel.

- Stakeholder roles — The supervisors were defined specific roles to replicate a real scenario.

- Design Phases — The two members discuss some design decisions in each of the V-model phase. The stakeholder roles are re-defined in every phase.

- Validation — The members use all the commands and test all the features of D3driver that includes working of "D3driver messaging extension", "configurable D3driver tab" , "search box", "sort button", "Export as PDF" button and "View JSON" button.

- At the end of this workshop, the members were also given a survey of questionnaire that was based on the evaluation criteria.

### 7.2.1        User stories

There are 3 V-model design phases that are taken into account by D3driver to document design decisions hence 3 scenarios are illustrated in this section. The name of the design phase and the roles defined are mentioned for each illustration. There are also member names present in the parenthesis according to their MS teams user IDs. The use case to each phase is also elaborated. To begin with, one of the members initialized the channel by invoking the bot as shown in Figure 7-1, an appropriate message sent by the bot can also be observed in the screenshot.
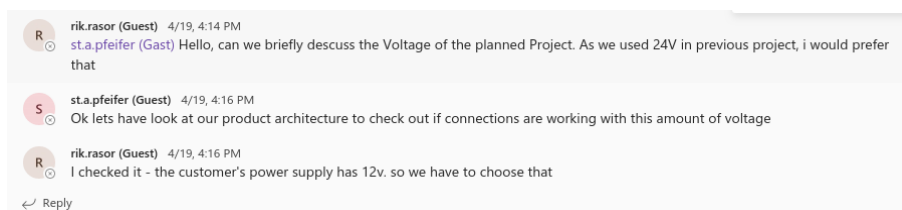
1. Phase - **Architecture phase**
   Roles - **Product owner (st.a.pfeifer (Gast)) and Systems architect (rik.rasor (Gast))**
   Use case 1 - **Decision on Logical design**

   In this phase, the two members discuss in short about a logical design related to Voltage as shown in Figure 7-2. These conversations are termed as normal messages
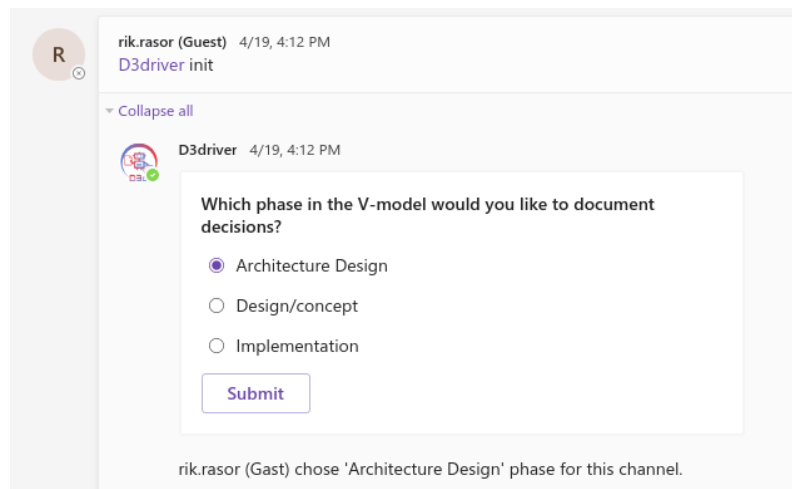
*Figure 7-1:     Channel initialization*



*Figure 7-2:     Normal group discussions*

in the group chat which are of least importance. After a quick initial discussion, the team members choose to document the design decision which they think is significant. Hence, they use D3driver messaging extension to document their decision. When they submit the decision to the group chat, a decision card is posted by D3driver and is depicted in Figure 7-3.
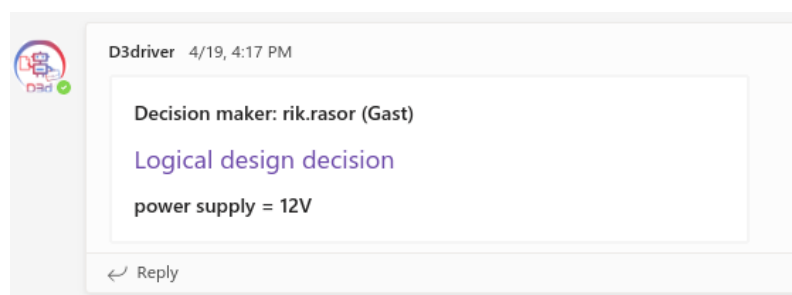


*Figure 7-3:     Logical design decision card*

Use case 2 - **Decision on Functional design**

In the subsequent use case, the members had a brief discussion on the functionality of intended device of their planned project. One of the members inquired if the device should be connected to an app or was it sufficient to use the control on the device itself. The other member chose their preference and made a decision and documented it using D3driver as shown in Figure 7-4
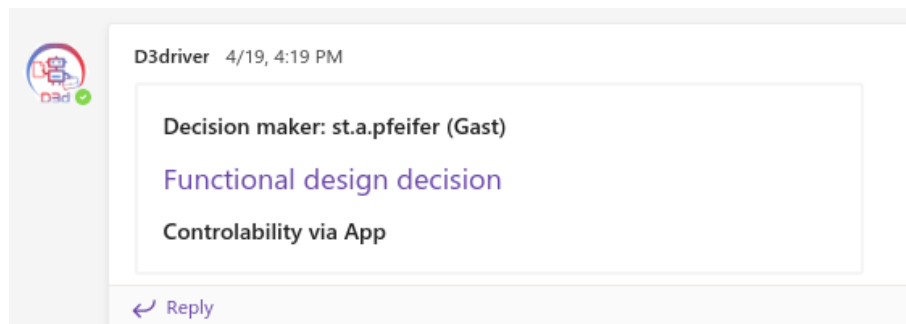


*Figure 7-4:     Functional design decision card*

The team members wished to re-initialize the channel with a different V-model phase. Hence, they saved their previous design decisions by clicking on the "Export as PDF" in D3driver tab as shown in Figure 7-5. The downloaded file was also viewed as shown in Figure 7-6. They members were also able to view the metadata by clicking on "View JSON" as shown in Figure 7-7. The next step was to use the delete command before re-initialization as shown in Figure 7-8
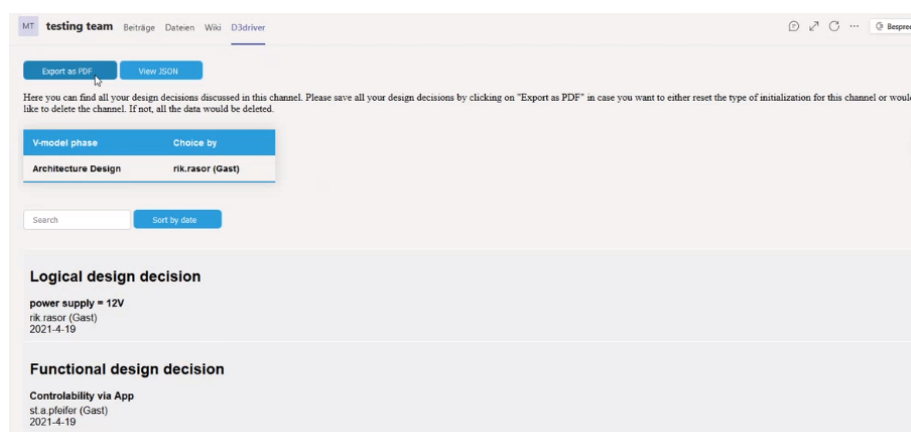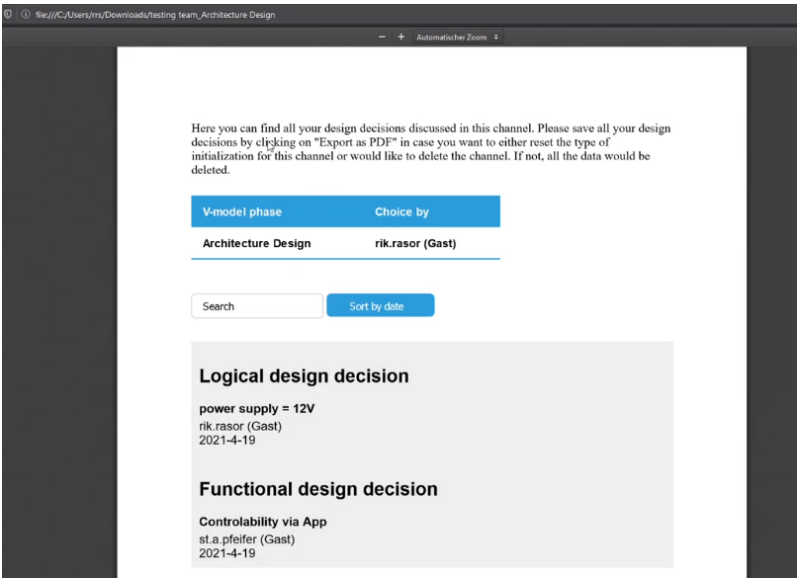


*Figure 7-5:     D3driver tab features*

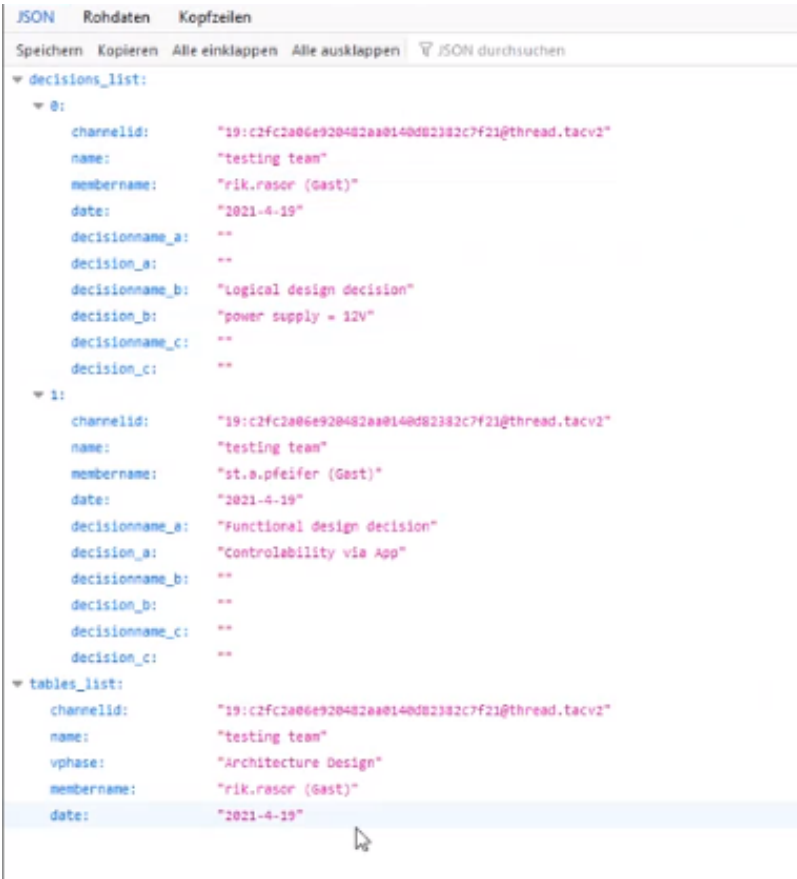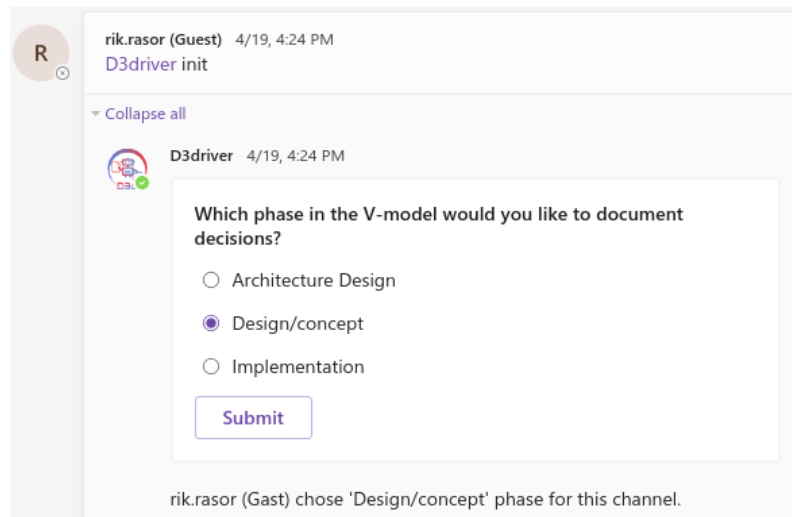*Figure 7-6:      View of downloaded PDF file*



*Figure 7-7:      JSON data view*

*Figure 7-8:     Delete command message*



*Figure 7-9:     Re-initialization of channel to Design/concept*

2. Phase - **Design/Concept phase**
   Roles - **Software Engineer (st.a.pfeifer (Gast) and Mechanical Engineer(rik.rasor(Guest))**
   Use case 3 - **Decision on Design alternatives**

   The members once again re-initialize the channel to Design/Concept as shown in Figure 7-9. The software engineer discusses control-ability and says that control through software is one option and a motor controller can be implemented so as to regulate the momentum and the velocity of the drives. On the other hand, a mechanical engineer mentions that the control-ability through a clutch is also a good idea. In this scenario, the team members after a discussion arrive at a design decision that they want to document. Hence, they once again use D3driver for decision documentation as shown in Figure 7-10.

3. Phase - **Implementation phase**
   Roles - **Two Software engineers (st.a.pfeifer (Gast) and rik.rasor (Gast))**
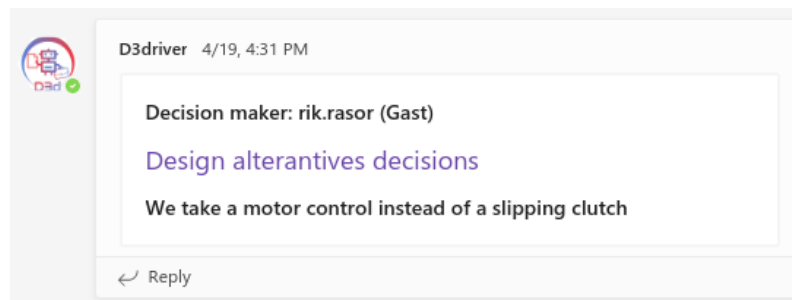   Use case 4 - **Decisions on software design**

*Figure 7-10:     Design alternative decision card*

At this point, the team saves the discussed decisions, deletes the current initialization and finally re-initializes for the last time as shown in Figure 7-11 to discuss implementation design decisions. The discussion during the implementation phase is related to the controller parameters. The software implementation of a Proportional–Integral–Derivative (PID) controller for the velocity control of the system was discussed. The two software engineers examine the better values to "P" and finally an experienced decision is made and the team once again uses D3driver to document the software design decision as shown in Figure 7-12. As a final step, the team also utilizes the search box feature as shown in Figure 7-13. The sort button was also reviewed and the dates can be sorted in either ascending or descending order as shown in Figure 7-14

At this stage, the validation is said to be complete. It can be implied from the user stories and screenshots of the simulation that the D3driver behaves as expected and has proved it's potential to carry out all it's tasks adequately well. The two stakeholders(supervisors) undertook a survey that sought their user experience of D3driver. The survey questionnaire was framed based on the evaluation criteria. The survey participants are also in the view that the bot performs as anticipated and their responses show that D3driver is a winner in managing and documenting design decisions. Their responses have been recorded and can be viewed here[1] for reference.

---

[1]https://docs.google.com/forms/d/1U45XPGRnw0L9-CVKsVAX0R-FlWS4dMBfwYolZOzlVZE/viewanalytics
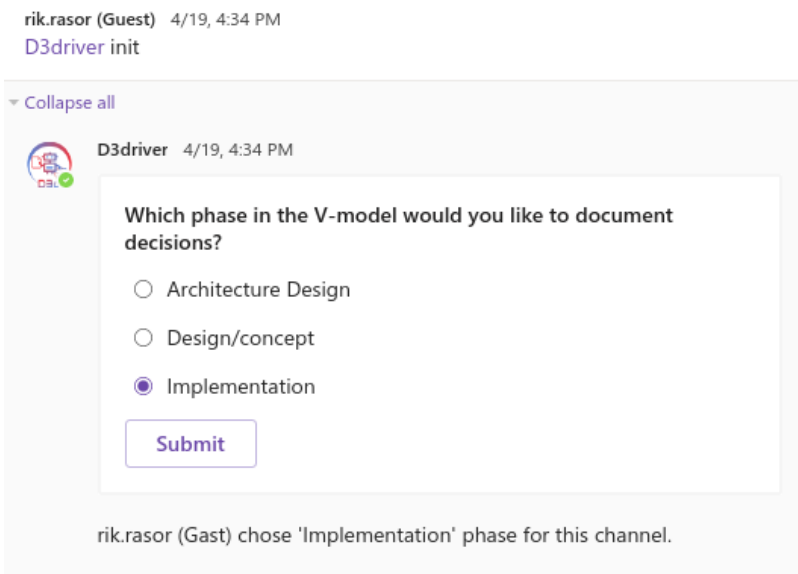
*Figure 7-11:     Channel initialized to Implementation*
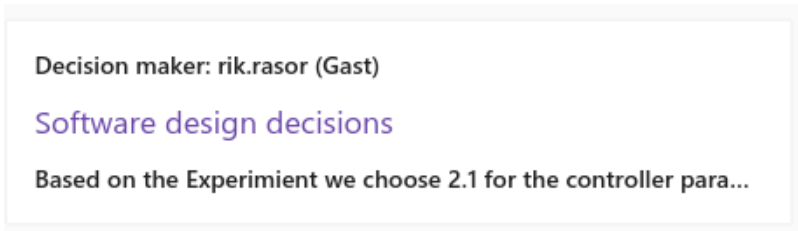


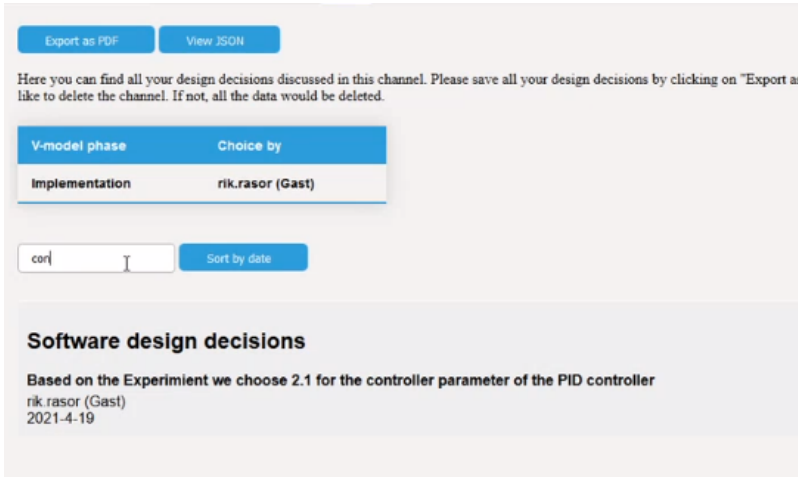*Figure 7-12:     Software design decision card*
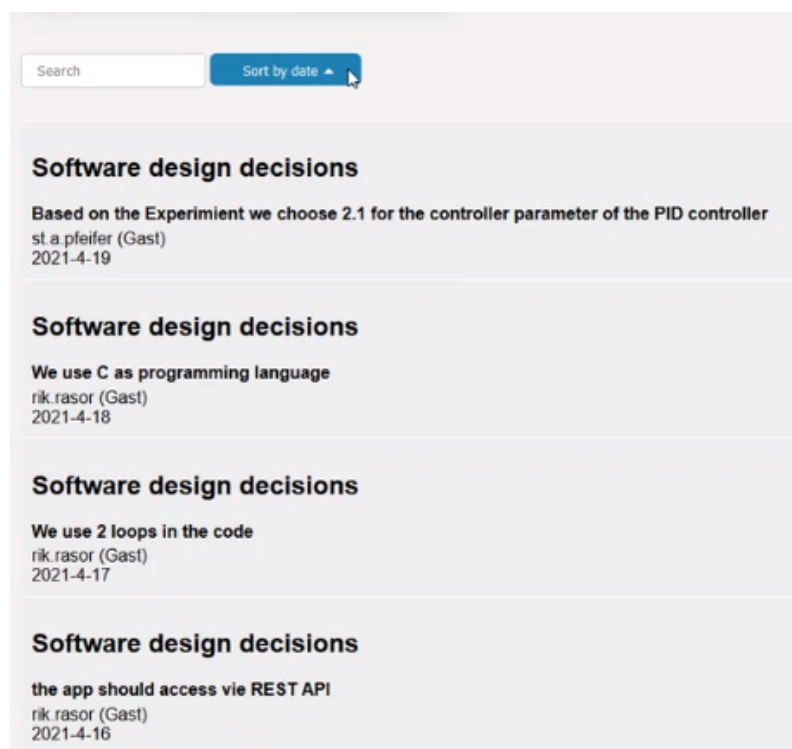


*Figure 7-13:     Search box*

*Figure 7-14:      Sort button*

## 8        Conclusion and Future Enhancements

In conclusion, D3driver has been successful in reaching the goals defined for this thesis implementation. D3driver was meant to ease the design decision making process for a multi-disciplinary team like mechatronics and the bot renders all the needful functionalities to accomplish the targets.

The thesis embarked with a notion to stress the emphasis of employee motivation especially towards documentation during a collective chat discussion in an inter-disciplinary team. Although, there were two other objectives put forward for the sake of employee motivation during the proposal phase, the scope and time limitation of the thesis had to be considered. The objective with the highest priority has been successfully implementing in the course of this thesis. The thesis in it's initial phases established the fact that collaboration is an essential component of any team especially when there are members from various domains. Teamwork, partnership and co-ordination normally happens over a collaboration tool and if that serves as an important aspect, then documenting chief facts and observations in the process is also equally important. The importance of documentation in mechatronics was discovered in parallel. At that moment, there was no system in place to handle the documentation functionality as per the need. Henceforth, it made sense to define what is lacking and what needs to be done further. This was followed by defining a problem statement. Once the problem was described, objectives and solution ideas were aligned and therefore the mission was to develop a bot preferably in MS Teams to motivate users upon documenting design decisions of mechatronics in a group chat. To this end, the background study began. The literature survey of mechatronics systems, processes, collaboration tools, decision making in mechatronics and bots in MS teams helped to channelize the research and implementation in the right direction. Thereafter, four prominent stages broke out which is as follows.

At this point, this thesis work had the right foundation to go ahead with the first stage of evaluating existing solutions to the described problem. This phase was more of a research task to determine the pros and cons of already existing bots in a couple of known collaboration tools. This involved establishing a set of evaluation criteria based upon the requirements and objectives of this thesis work. The bots were installed and tested to see which evaluation criteria was fulfilled, followed up a summary report. This stage formed a great groundwork for further stages.

The next two stages, design and implementation of a motivational bot represent the crux of this work. After some study on what technologies and applications would best suit the thesis specifications, the name, logo, architecture, workflow and approaches to implementation originated and the same has been explained. During the implementation stage, bot development guidance provided by official Microsoft documentation helped in prototyping D3driver and it's features. A detailed report on D3driver's installation, usage, components, intricate working has been presented. In addition, interested users can also take a look at the GitHub repository[1] for source codes and user manual.

The final stage in this work was the verification and validation of D3driver. This stage first verifies to see if the bot meets all the required specifications and confirms that the bot can be made available to the users. D3driver was reviewed against the same set evaluation criteria that was established to review existing bots. Next, validation was done by having two members from different domains simulate engineering scenarios where design decision documentation was done with the use of D3driver. The results of this demonstration was logged and submitted in this report. With this, all the four stages of bot prototyping has been victoriously completed.

## 8.1        Future enhancements

Future work on D3driver can persevere in 2 different paths. One in which D3driver's current functionalities can be upgraded and another direction would be to add new innovations (other than documentation functionality) altogether. The latter has already been proposed in Chapter 1. A clear conception on two more functionalities in the area of employee motivation, their problem description, objectives and solutions has also been suggested in Sections 1.1 , 1.2 and 1.3 respectively. These two were marked as secondary or optional goals in this thesis.

For the purpose of improving the current features of D3driver, the following points can be noted.

- Introducing another option to the list of initialization options. Right now, there are 3 phases of V-model as initialization choices, however there could be a "custom" choice and introducing "custom" sub-options. With this, the users will be able to label their own decision names and will be able to document anything other than just design decisions.

---

[1]https://github.com/bhargavimohan/Masterthesis_bot2021

- Designing graphs and statistics in the D3driver tab to see the total number of decisions, highest number of decisions taken by a particular member, highest number of decision types discussed and a few more useful facts and figures depending on the actual requirement in a team.

- The orientation of the decisions displayed in the D3driver tab can be visualized in a more refined way. There can be different layouts for each decision type with their own search and sort functionalities. This would further make the search feature optimized when there are thousands of decisions discussed.

- The sort functionality can also be further enhanced by introducing more sort by options like *sort by members* and *sort by decision type*. This would help users to generate a precise report of decision types or decision makers. For instance, *sort by members* functionality would line up all the decisions taken by a particular member and when this is on display, the users can use "Export as PDF" button to generate this report before it is refreshed.

- At present, D3driver can be added to any channel. But this can be extended to "*Add to meeting*" to document decisions pertaining to V-model design phases while the meeting is going on. This can help the users maintain a separate thread of design decision discussions exclusively during meetings.

- The database structure can be made more advanced by using a more sophisticated database other than TinyDB to handle huge records to enable scalability in production environment.

- Finally, making D3driver a production-ready bot to deploy it in Microsoft azure is a major upgrade. This will result in an implicit performance upgrade of not having to use *ngrok* technology and can avoid frequent *ngrok URL* update in the code. This would also make the bot be available to public for usage.

Apart from the above ideas, there are some minor design advancements to be considered:

- A welcome card with greetings and detailed instructions on the available commands and their usage can be designed to be sent on bot start up without any user requiring to invoke the bot.

- A special card asking deletion confirmation can be designed after using the *"del"* command. Currently, the "del" command would erase all the data when used without a warning.

- The current version of adaptive cards in MS teams has a limitation on the radio button graphical appearance. It is visible only to the user making an option during channel initialization and the radio button appearance is not visible to other members who aren't making the choice (however, appropriate messages appear immediately on the choices made for users who do not make the initialization choice). Hence, when an update on the version of adaptive card is available, changes to reflect this design improvement can be considered.

# 9    Bibliography

[ABS]      ABDELLATIF, A.; BADRAN, K. ; SHIHAB, E.:   MSRBot: Using bots to
           answer questions from software repositories. 25, Nr. 3, 1834–1863. `http://dx.doi.org/10.1007/s10664-019-09788-5`. – DOI 10.1007/s10664–
           019–09788–5. – ISSN 1382–3256, 1573–7616

[ACFT$^+$] ALVAREZ CABRERA, A.; FOEKEN, M.; TEKIN, O.; WOESTENENK, K.;
           ERDEN, M.; DE SCHUTTER, B.; TOOREN, M. van; BABUŠKA, R.;
           HOUTEN, F. van ; TOMIYAMA, T.:   Towards automation of control
           software: A review of challenges in mechatronic design. 20, Nr. 8, 876–886.
           `http://dx.doi.org/10.1016/j.mechatronics.2010.05.003`. –   DOI
           10.1016/j.mechatronics.2010.05.003. – ISSN 09574158

[Bra04]    BRADLEY, D.:     What is Mechatronics and Why Teach it?     In:
           *International Journal of Electrical Engineering &amp;Education* 41 (2004),
           Nr. 4, S. 275–291. `http://dx.doi.org/10.7227/ijeee.41.4.2`. – DOI
           10.7227/ijeee.41.4.2

[bri10]    *The History of Mechatronics*. `https://www.brighthubengineering.com/manufacturing-technology/71180-the-history-of-mechatronics/`.
           Version: May 2010

[BVK$^+$16] BITZER, M.; VIELHABER, M.; KASPAR, J. u. a.:     Product Lifecycle
           Management–How to adapt PLM to support changing product development
           processes in industry? In: *DS 85-1: Proceedings of NordDesign 2016, Volume
           1, Trondheim, Norway, 10th-12th August 2016* (2016), S. 360–369

[DHD17]    DIXIT, U. S.; HAZARIKA, M. ; DAVIM, J. P.: History of mechatronics. In: *A
           brief history of mechanical engineering.* Springer, 2017, S. 147–164

[GHB$^+$18] GRÄSSLER, I.; HENTZE, J.; BRUCKMANN, T. u. a.:     V-models for
           interdisciplinary systems engineering. In: *DS 92: Proceedings of the DESIGN
           2018 15th International Design Conference*, 2018, S. 747–756

[GW19]      GILSON, F.; WEYNS, D.:  When natural language processing jumps into
            collaborative software engineering. In: *2019 IEEE International Conference
            on Software Architecture Companion (ICSA-C)* IEEE, 2019, S. 238–241

[Haba]      HABIB, M. K.: Mechatronics Engineering The Evolution, the Needs and the
            Challenges. In: *IECON 2006 - 32nd Annual Conference on IEEE Industrial
            Electronics*, S. 4510–4515. – ISSN: 1553-572X

[Habb]      HABIB, M.:  Mechatronics - A unifying interdisciplinary and intelligent
            engineering science paradigm. 1, Nr. 2, 12–24. `http://dx.doi.org/10.`
            `1109/MIE.2007.901480`. – DOI 10.1109/MIE.2007.901480. – ISSN 1932–
            4529

[HAT]       HEESCH, U. van; AVGERIOU, P. ; TANG, A.: Does decision documentation
            help junior designers rationalize their decisions? A comparative multiple-case
            study. 86, Nr. 6, 1545–1565. `http://dx.doi.org/10.1016/j.jss.2013.`
            `01.057`. – DOI 10.1016/j.jss.2013.01.057. – ISSN 01641212

[HB]        HUBBARD, M.; BAILEY, M. J.:  *Mastering Microsoft Teams: End User
            Guide to Practical Usage, Collaboration, and Governance.* Apress. `http:`
            `//dx.doi.org/10.1007/978-1-4842-3670-3`. `http://dx.doi.org/10.`
            `1007/978-1-4842-3670-3`. –  ISBN 978–1–4842–3669–7 978–1–4842–
            3670–3

[Her]       HERSH, M.: Sustainable decision making: the role of decision support systems.
            29, Nr. 3, S. 395–408. `http://dx.doi.org/10.1109/5326.777075`. – DOI
            10.1109/5326.777075. –  ISSN 1558–2442. –  Conference Name: IEEE
            Transactions on Systems, Man, and Cybernetics, Part C (Applications and
            Reviews)

[HJC+]      HAMIDA, S. B.; JANKOVIC, M.; CALLOT, M.; MONCEAUX, A. ; ECKERT,
            C.: TOWARDS A DECISION SUPPORT FRAMEWORK FOR SYSTEM
            ARCHITECTURE DESIGN.

[IR]        IRSHORN, H.; R., S.: *NASA SYSTEMS ENGINEERING HANDBOOK.* S.
            160–171

[Kas]       KASSER, J.:      IMPROVING THE SYSTEMS ENGINEERING
            DOCUMENTATION PRODUCTION PROCESS.

[LBP]      LOMAS, C.; BURKE, M. ; PAGE, C. L.: Collaboration Tools.

[LSZ]      LEBEUF, C.; STOREY, M.-A. ; ZAGALSKY, A.: Software Bots. 35, Nr.
           1, S. 18–23. http://dx.doi.org/10.1109/MS.2017.4541027. – DOI
           10.1109/MS.2017.4541027. – ISSN 1937–4194. – Conference Name: IEEE
           Software

[LZFS]     LEBEUF, C.; ZAGALSKY, A.; FOUCAULT, M. ; S, M.-A.: Defining and
           Classifying Software Bots: A Faceted Taxonomy. In: *2019 IEEE/ACM 1st
           International Workshop on Bots in Software Engineering (BotSE)*, S. 1–6

[LZS]      LEBEUF, C.; ZAGALSKY, A. ; STOREY, M.-A.: How Software Developers
           Mitigate Collaboration Friction with Chatbots.

[LZSS]     LIN, B.; ZAGALSKY, A. E.; STOREY, M.-A. ; SEREBRENIK, A.: Why
           Developers Are Slacking Off: Understanding How Software Teams Use
           Slack. In: *Proceedings of the 19th ACM Conference on Computer Supported
           Cooperative Work and Social Computing Companion - CSCW '16 Companion*,
           ACM Press. – ISBN 978–1–4503–3950–6, 333–336

[MAH$^+$]  MCHAREK, M.; AZIB, T.; HAMMADI, M.; CHOLEY, J.-Y. ; LAROUCI, C.:
           Knowledge sharing for mechatronic systems design and optimization. 51, Nr.
           11, 1365–1370. http://dx.doi.org/10.1016/j.ifacol.2018.08.338. –
           DOI 10.1016/j.ifacol.2018.08.338. – ISSN 24058963

[MBP]      MONTANI, F.; BOUDRIAS, J.-S. ; PIGEON, M.: Employee recognition,
           meaningfulness and behavioural involvement: test of a moderated mediation
           model. 31, Nr. 3, 356–384. http://dx.doi.org/10.1080/09585192.
           2017.1288153. – DOI 10.1080/09585192.2017.1288153. – ISSN 0958–
           5192, 1466–4399

[Moh19]    MOHAN, R.: The Chat bot revolution and the Indian HR Professionals. In:
           *International Journal of Information and Computing Science* 6 (2019), Nr. 3,
           S. 489–499

[MP]       MURESAN, A.; POHL, H.: Chats with Bots: Balancing Imitation and
           Engagement. In: *Extended Abstracts of the 2019 CHI Conference on Human
           Factors in Computing Systems*, ACM. – ISBN 978–1–4503–5971–9, 1–6

[MTA$^+$]   MANTEUFFEL, C.; TOFAN, D.; AVGERIOU, P.; KOZIOLEK, H. ;
           GOLDSCHMIDT, T.:  Decision architect – A decision documentation tool
           for industry. 112, 181–198. `http://dx.doi.org/10.1016/j.jss.2015.`
           `10.034`. – DOI 10.1016/j.jss.2015.10.034. – ISSN 01641212

[Neu]      NEUMANN, F.:        MECHATRONIC PRODUCT DEVELOPMENT:
           POTENTIALS, CHALLENGES, TERMINOLOGY.

[RBA17]    RAD, B. B.; BHATTI, H. J. ; AHMADI, M.:  An introduction to docker and
           analysis of its performance. In: *International Journal of Computer Science
           and Network Security (IJCSNS)* 17 (2017), Nr. 3, S. 228

[SEH]      SURI, V. K.; ELIA, M. ; HILLEGERSBERG, J. van: Software Bots - The Next
           Frontier for Shared Services and Functional Excellence. `http://dx.doi.`
           `org/10.1007/978-3-319-70305-3_5`. In: OSHRI, I. (Hrsg.); KOTLARSKY,
           J. (Hrsg.) ; WILLCOCKS, L. P. (Hrsg.): *Global Sourcing of Digital Services:
           Micro and Macro Perspectives* Bd. 306. Springer International Publishing. –
           DOI 10.1007/978–3–319–70305–3_5. – ISBN 978–3–319–70304–6 978–3–
           319–70305–3, 81–94. – Series Title: Lecture Notes in Business Information
           Processing

[SSR$^+$]   STOREY, M.-A.; SEREBRENIK, A.; ROSÉ, C. P.; ZIMMERMANN, T. ;
           HERBSLEB, J. D.: BOTse: Bots in Software Engineering.

[Thr]      THRAMBOULIDIS, K.:  Challenges in the development of Mechatronic
           systems: The Mechatronic Component.  In: *2008 IEEE International
           Conference on Emerging Technologies and Factory Automation*, S. 624–631. –
           ISSN: 1946-0759

[WE]       WALDEN, D. D.; ESEP:, G. J. R. E. K. J. F. E. R. D. H. T. M. S. C.: *INCOSE
           SYSTEMS ENGINEERING HANDBOOK.* S. 110–114

[WTH]      WIKANDER, J.; TORNGREN, M. ; HANSON, M.: The science and education
           of mechatronics engineering. 8, Nr. 2, S. 20–26. `http://dx.doi.org/`
           `10.1109/100.932753`. – DOI 10.1109/100.932753. – ISSN 1558–223X. –
           Conference Name: IEEE Robotics Automation Magazine

[ZC]       ZHANG, A. X.; CRANSHAW, J.:  Making Sense of Group Chat through
           Collaborative Tagging and Summarization. 2, 1–27. `http://dx.doi.org/`

10.1145/3274465. – DOI 10.1145/3274465. – ISSN 2573–0142, 2573–0142

[ZH]        ZHENGMENG, C.; HAOXIANG, J.:   A brief review on Decision Support Systems and it's applications. In: *2011 IEEE International Symposium on IT in Medicine and Education* Bd. 2, S. 401–405