# Unified Modeling Language (UML) and Software Design Specification (SDS)

Hayden Eubanks

September 24, 2023

## Table of Contents

## Introduction

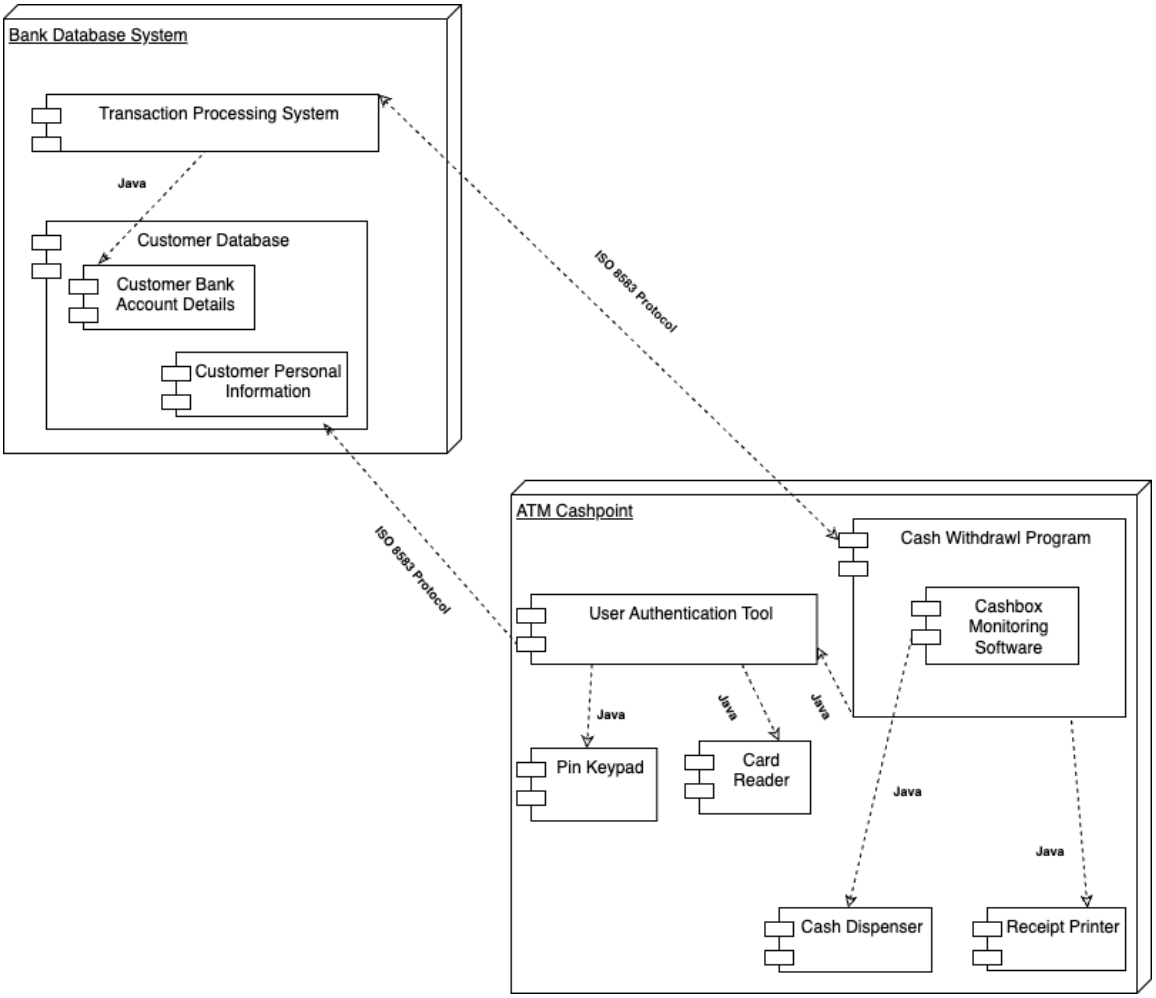
One factor that is crucial to the success of a development project is that an adequate level of design is performed (Tsui, Karam, & Bernal, 2016), and as such developers have sought out design patterns and methodologies that increase the chances of a project succeeding. In addition to this, development errors are commonly introduced early into the software development lifecycle (SDLC), and without proper design, the lack of unified direction in development could compound the quantity of these errors (Fernández-Sáez et al., 2018). Solid design methodologies can then be seen to assist development by both facilitating a unified approach among the development team and allowing errors to be spotted earlier in development where they are easier and less costly to fix (Tsui, Karam, & Bernal, 2016). To facilitate the shared understanding of a project's design, several methodologies such as unified modeling language (UML) diagrams have been developed. By examining UML models such as deployment, sequence, navigational hierarchy, and data flow diagrams a developer can then better understand the relationship between components and how data flows between them (Fernández-Sáez et al., 2018).

UML diagrams are a powerful tool in the design and implementation of software as they provide several views of the software and interacting components (Koç et al., 2021). These views then enable a development team to examine software within the context of a specific level of detail and in doing so model a layer of abstraction by which design can be implemented (Ozkaya & Erata, 2020). One specific aspect of design that UML diagrams are capable of modeling is the relationship between system components and the use of deployment, sequence, navigational hierarchy, and data flow diagrams can vastly assist in providing a shared view of this interaction among the development team. These views can

highlight system design considerations such as the protocols used in communication between components, the physical arrangement of components, the sequence in which components interact, and the flow of data throughout a system (Ozkaya & Erata, 2020). This information is essential for a developer to understand as errors in these components could result in severe security or project quality implications (Koç et al., 2021). The potential cost of these errors then highlights the importance of modeling the project design architecture (Ozkaya & Erata, 2020), and through examining the UML diagrams present in this design specification document, an understanding of the architecture of the cash withdraw function of an ATM can be realized.

Throughout this design specification, an analysis of the withdrawal function of an ATM will be analyzed and through this analysis, an understanding of the interactions between the customer, the ATM, and the bank database can be described. The UML diagrams representing this functionality will then expand upon the relationship between the components to represent where the components exist concerning each other, the protocols used to communicate between the components, the sequence in which the components interact, the interfaces portrayed to the user at any given state of the process, and the flow of data between components. Each of these factors of component relationship is essential to the success of the withdraw function and as such, great care has been taken to thoroughly model the respective layer to be portrayed. However, to fulfill the goals of abstraction in modeling diagrams for each view, only the relevant components for each view were included and this abstraction can then allow developers to work with only the relevant components within a development domain (Tsui, Karam, & Bernal, 2016).

1. **Deployment Diagram: Withdraw Cash From ATM**

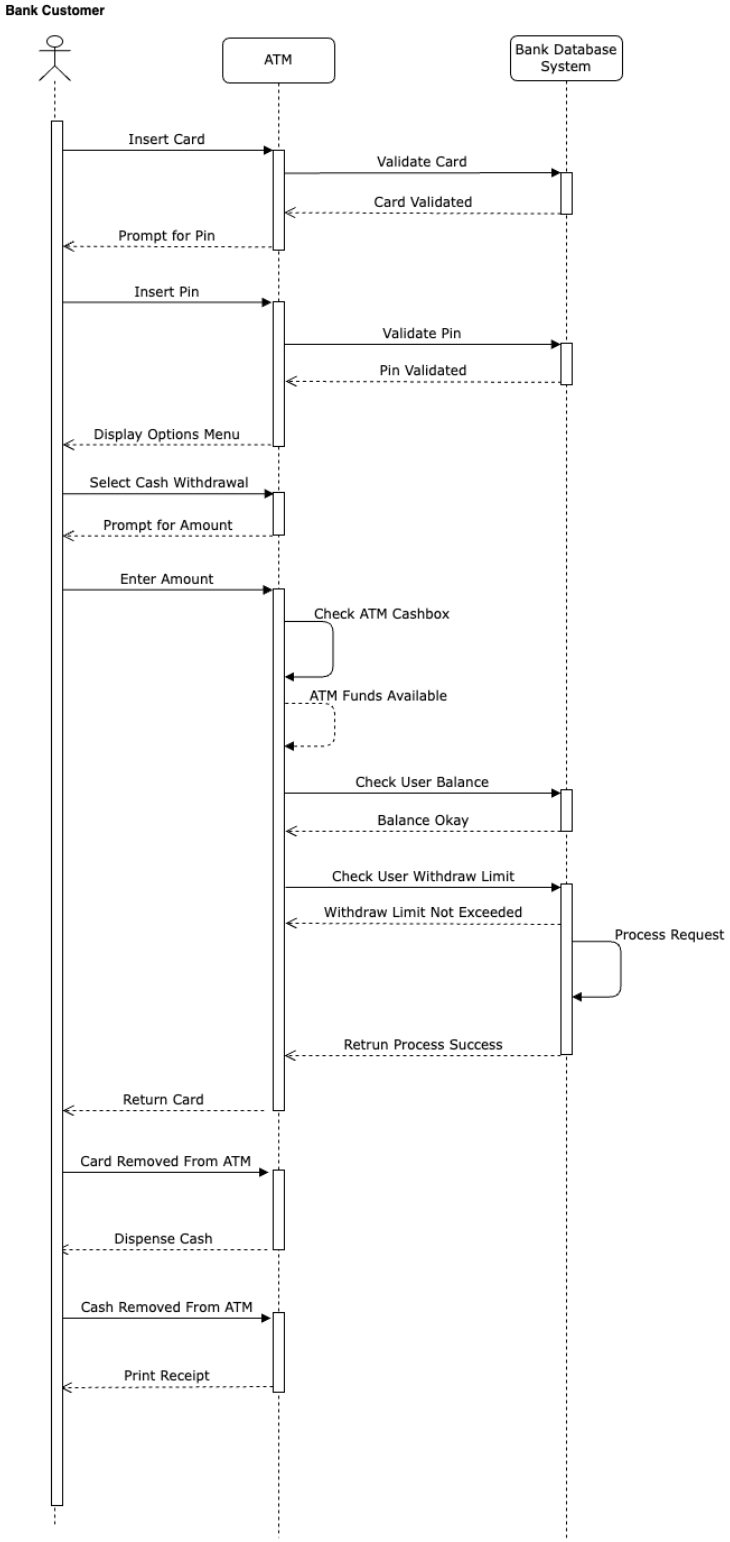



The first diagram to be examined is the deployment diagram which models the architecture of software components and their relationship to each other through subcomponent communication (Microsoft, 2021b). Within this diagram, each primary piece of software can be represented as a node (University of Pittsburgh, n.d.), and in this instance, the two primary nodes can be seen as the ATM cashpoint software and the bank's customer database system. Each of these software artifacts then contains subcomponents of processes that run as part of the software node. On the bank database system, these subcomponents include the backend components of the transaction processing software as well as the database of customer information. On the ATM

software, components such as the user authentication process, the cash withdrawal program, and software managing the hardware components can be seen. When components are interacting within the same software node, the Java language is used to run code that manages the interactions. However, data being sent over the network to external components must be thoroughly protected and for this reason, the ISO 8583 Protocol, which is standard in the transportation of ATM transaction data, will be used (IBM, 2023).

The deployment diagram is a valuable UML artifact as it allows developers to visualize a system's software components as well as how they are arranged within the system architecture (Microsoft, 2021b). Further, this view also provides the basic understanding of subcomponent arrangement and with this, models subcomponent interaction. This understanding is then valuable in design as it allows developers to focus development efforts within a software node while ensuring that the functionality for communication exists where required (Microsoft, 2021b). This can then allow for the reduction of errors due to missed functionality implementation and in doing so increase the security and resiliency of the project deliverable.

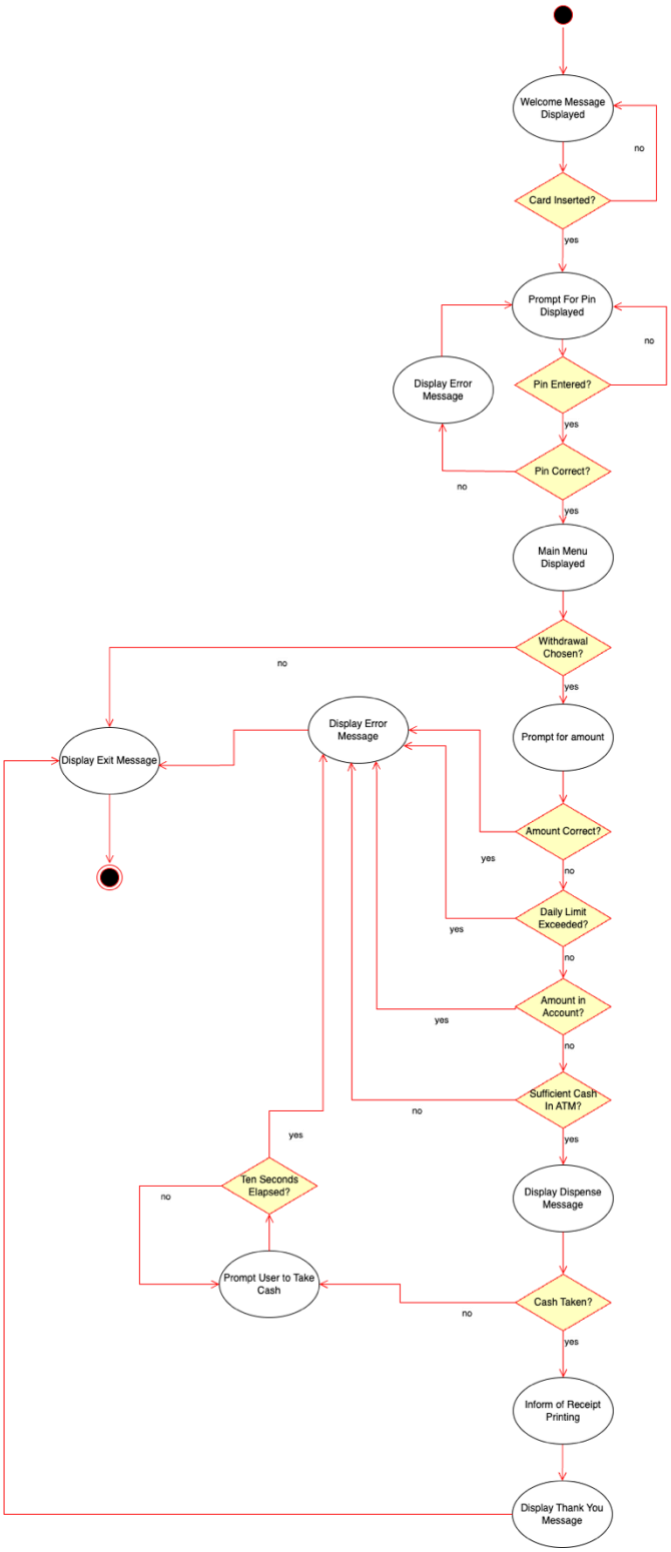
## 2. Sequence Diagram



The next diagram, the sequence diagram, can then be used to model the sequence in which component interactions will occur (Microsoft, 2021a). As an extension of modeling these interactions, a view of which components are involved in a given task can be easily observed along with the relative time a system is engaged with a given task. In addition to the element of time, another factor that distinguishes the sequence diagram is the inclusion of the bank customer actor and how this customer interacts with the software systems. Observing the sequence diagram tasks can then allow a developer to understand the task being requested of each system as well as the value being returned to the calling component when the task is finished (Microsoft, 2021a). Understanding the relationship between system components over the time a task is being performed can then allow developers to develop software according to the task flow outlined in the diagram and ensure tasks are performed by each component when required.

Within the context of the ATM withdrawal feature, the components include the previously mentioned ATM software and bank database in addition to the customer interacting with those systems. The customer object can be seen to interact with the ATM interface by taking actions such as inserting their card or entering their PIN. When the customer takes these actions, the ATM software performs tasks both natively within the software as well as making function calls to the database when needed. Through this model, the ATM can be seen to function as an interface between the customer and the database and as such, the sequence diagram provides a starting point from which further diagrams such as the navigational hierarchy can be created to model the interface components.

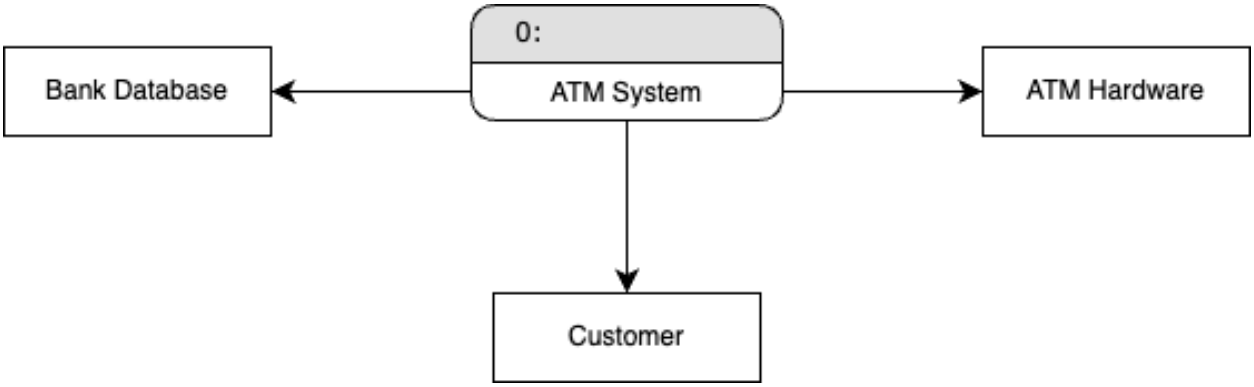
3. **Navigational Hierarchy**



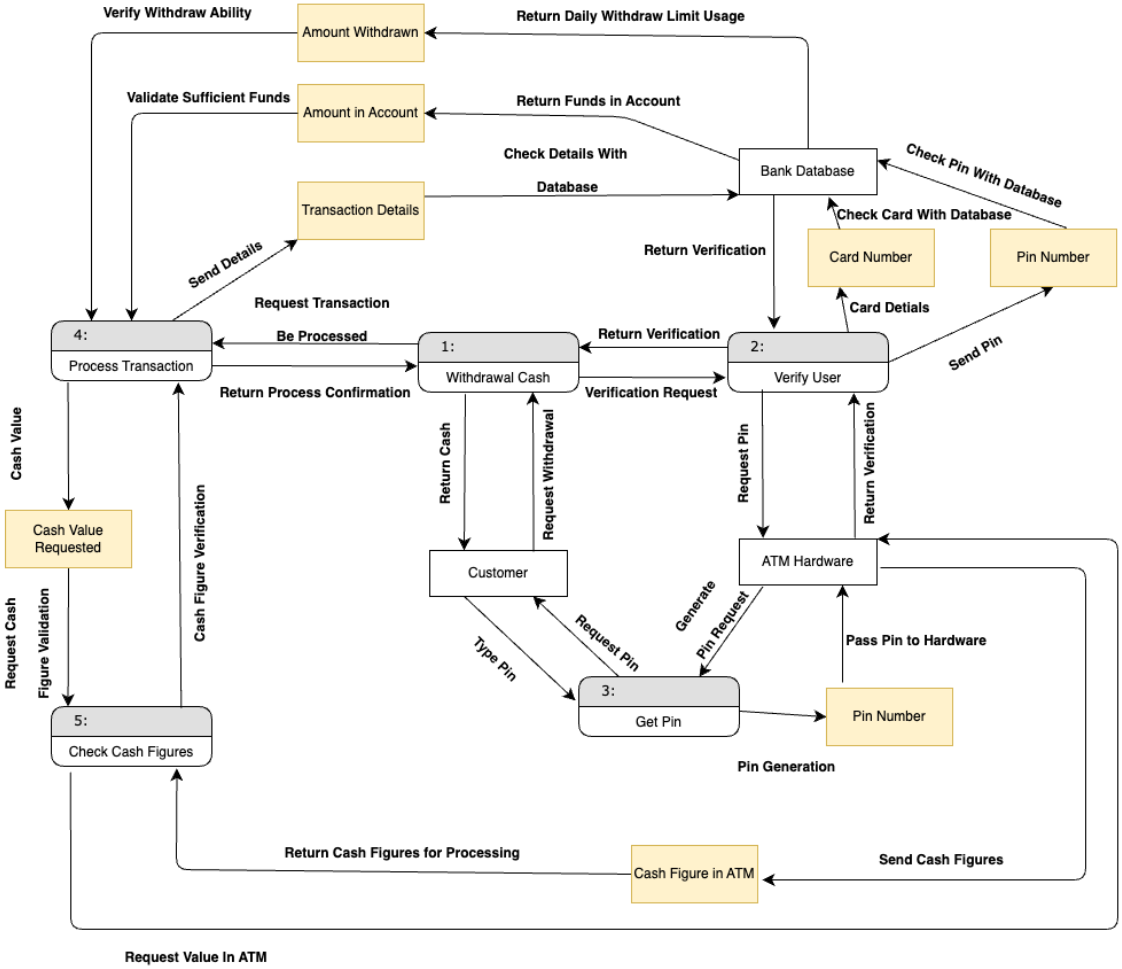
The navigational hierarchy diagram is similar to the activity diagram which models an activity's flow, but instead of modeling activity flow the diagram is used to model a user's interaction with interface components (Wong, 2020). As a user interacts with the system, the user will be navigating through interface elements that change based on the state of the system. For this reason, a navigational hierarchy diagram is used to model the change in interface elements as the user alters the state of the system through interaction. These interface changes can include prompting the user for specific types of input, displaying menus, or outputting messages to inform the user of process status. With this in mind, the flow of interface elements can then be seen as altered by user input, system status, or process variables highlighting the elements that developers must keep in mind when developing the transitions of interface states.

Within the context of the ATM use case, the interface elements portrayed to the user can be seen as represented through messages printed to the screen for the user to read. The user is then able to respond to these messages to navigate through the withdrawal process and with this the interface menus. Each of the interface menus can then be seen to inform the user of the appropriate actions to take next to progress the state of the process. If the user is non-compliant with these instructions, the navigational hierarchy model then portrays the interface changes to display error messages and allow the user to correct the problem or exit the program. As the user will primarily interact with the interface menus provided, great care must be placed on their design to ensure a good user experience (Hartson & Pyla, 2012). Navigational hierarchy diagrams can then be seen to greatly serve the development process as they allow developers to design interface elements with an understanding of the overarching interface flow.

4. **Data Flow Diagram Level 0**




## Data Flow Diagram Level 1



Another software development domain that is essential for developers to consider is the transfer of data throughout a system as well as the processes that engage the transfer of data (University of Florida, n.d.). This view can be provided through the UML diagram type of data flow diagrams and with this, data flow diagrams can be portrayed in increasing levels of abstraction. At the most basic level, data flow diagrams provide the most general view of a process, highlighting the components that interact with that process for the transfer of data. This most generalized level of abstraction is referred to as a level zero diagram (Tsui, Karam, & Bernal, 2016) as it provides only the most generalized view of system components without highlighting any data transfer itself. The next level of abstraction, level one, can be seen to portray the same system in more detail while highlighting major processes that request data and the actual data transit paths themselves. This view can then be seen to allow a developer to track data flow regarding a calling process and easily follow the path of data through the system. The protection of data is an essential aspect of a software system (Tsui, Karam, & Bernal, 2016), and understanding the flow of data through the system is essential to providing adequate data protection highlighting the importance of tools such as data flow diagrams in the SDLC.

Within the context of the ATM application, the flow of data can be tracked between each of the primary system entities. One important feature of this data flow is that no entity directly draws data from another entity, but instead, a process is used as an interface between entities and their data. This interface can greatly assist in data security as the intermediary process can enforce data protection controls (Tsui, Karam, & Bernal, 2016). Further, the level one diagram provides additional clarity for developers by highlighting function calls and returns so the direction of data flow can be observed in relation to its parent entity. This view allows

developers to track the flow of data throughout the ATM system and as such can allow developers to implement the appropriate data transfer functionality in the finished project.

## References


Fernández-Sáez, A. M., Chaudron, M. R. V., Genero, M., Institutionen för data- och informationsteknik, IT-fakulteten, Göteborgs universitet, Gothenburg University, Department of Computer Science and Engineering, & IT Faculty. (2018). An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles. *Empirical Software Engineering: An International Journal, 23*(6), 3281-3345. https://doi.org/10.1007/s10664-018-9599-4

Hartson, H. R., & Pyla, P. S. (2012). *The UX book: Process and guidelines for ensuring a quality user experience*. Morgan Kaufmann. https://doi.org/10.1016/C2010-0-66326-7

IBM. (June 21, 2023). *ISO 8583 messaging standard*. IBM. https://www.ibm.com/docs/en/integration-bus/10.0?topic=formats-iso8583-messaging-standard

Koç, H., Erdoğan, A. M., Barjakly, Y., & Peker, S. (March, 2021). UML diagrams in software engineering research: a systematic literature review. *Proceedings 74(1)*, 13. MDPI. https://www.mdpi.com/2504-3900/74/1/13

Microsoft. (2021a). *Create a UML sequence diagram*. Microsoft Support. https://support.microsoft.com/en-us/office/create-a-uml-sequence-diagram-c61c371b-b150-4958-b128-902000133b26

Microsoft. (2021b). Create a UML deployment diagram. Microsoft Support. https://support.microsoft.com/en-au/office/create-a-uml-deployment-diagram-ef282f3e-49a5-48f5-a6ae-69a6982a4543

Tsui, F, Karam, O., Bernal, B. (2016). *Essentials of Software Engineering* (4th ed.). Jones & Bartlett Learning. https://libertyonline.vitalsource.com/books/9781284129755

Ozkaya, M., & Erata, F. (2020). A survey on the practical use of UML for different software architecture viewpoints. *Information and Software Technology*, *121*, 106275. https://www.sciencedirect.com/science/article/abs/pii/S0950584920300252

University of Florida. (n.d.). *Creating an information system/ data flow diagram*. University of Florida. https://security.ufl.edu/resources/risk-assessment/creating-an-information-systemdata-flow-diagram/

University of Pittsburgh. (n.d.). *Deployment diagrams*. University of Pittsburgh. https://people.cs.pitt.edu/~chang/1635/c03UMLDPL/c02.htm

Wong, P. (February 25, 2020). *Navigation and architecture*. Indiana University. https://ux.iu.edu/writings/navigation-and-architecture/