12.20. Using Gane and Sarsen's technique, draw up a specification document for the Chocoholics Anonymous product described in Appendix A.

STEP 1: Data Flow Diagrams

Figure 1 – Logical Data flow diagram for maintaining the Chocoholic Anonymous product.

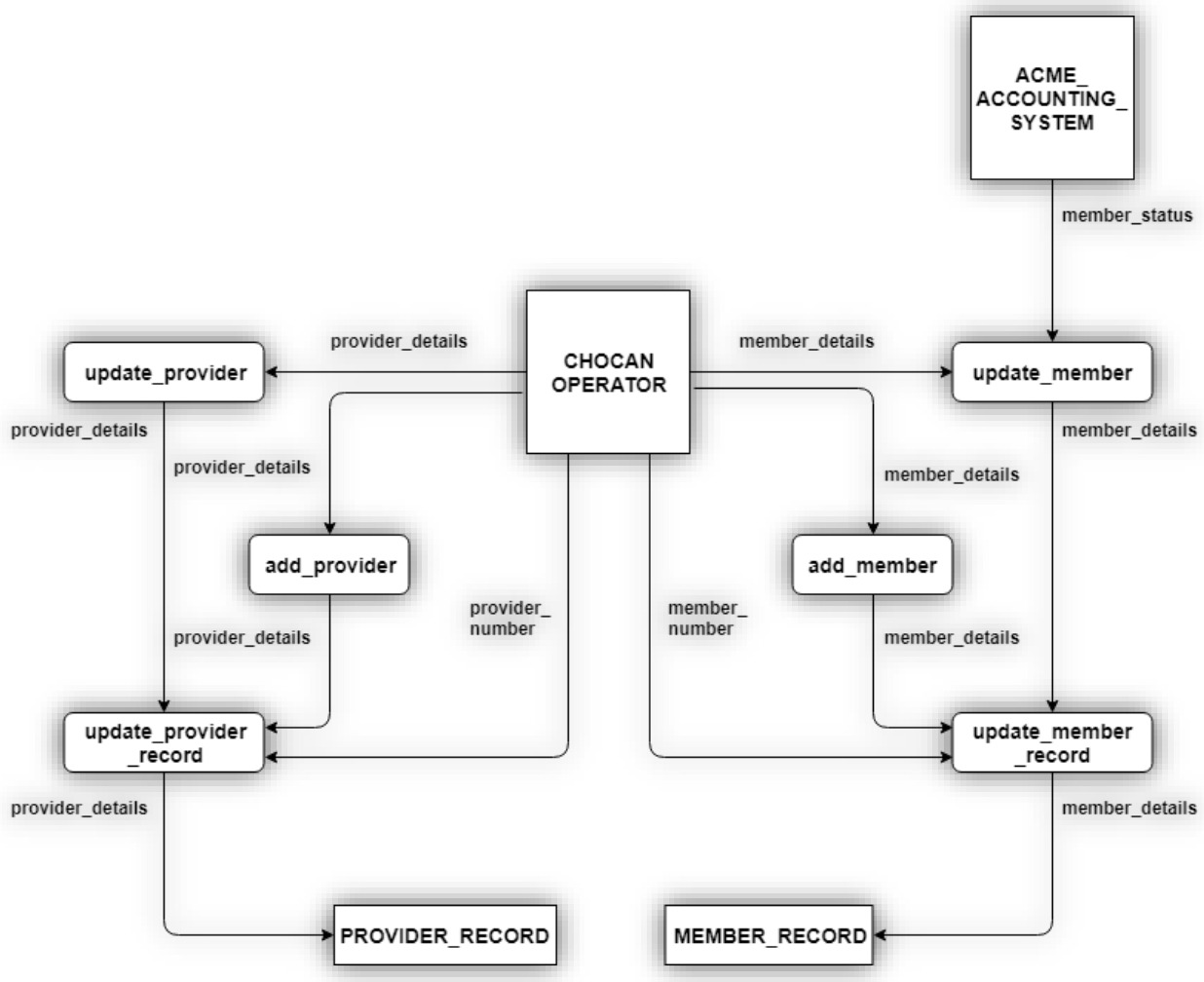


Figure 2 – Logical data flow diagram for report generation of the Chocoholic Anonymous product.

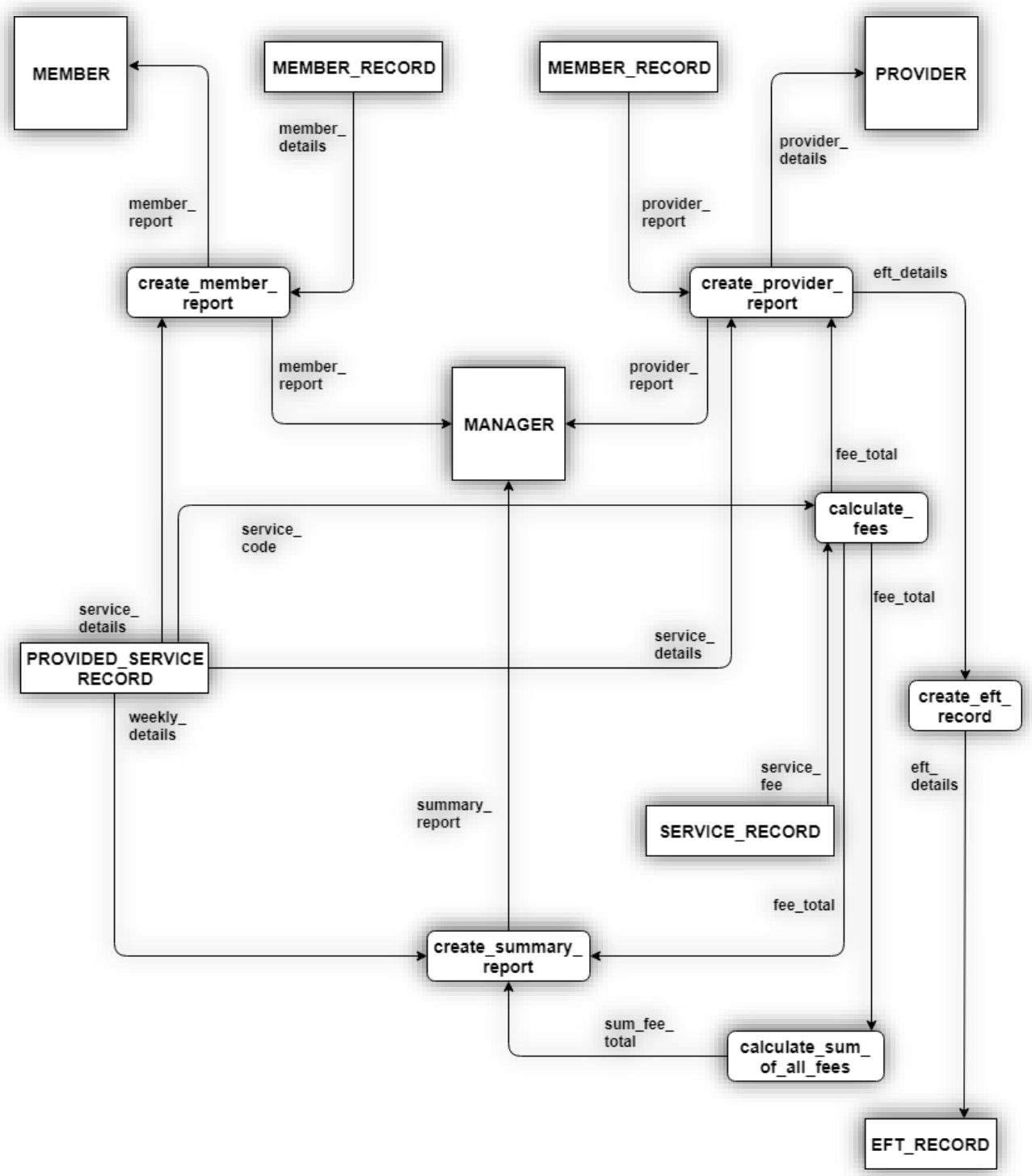
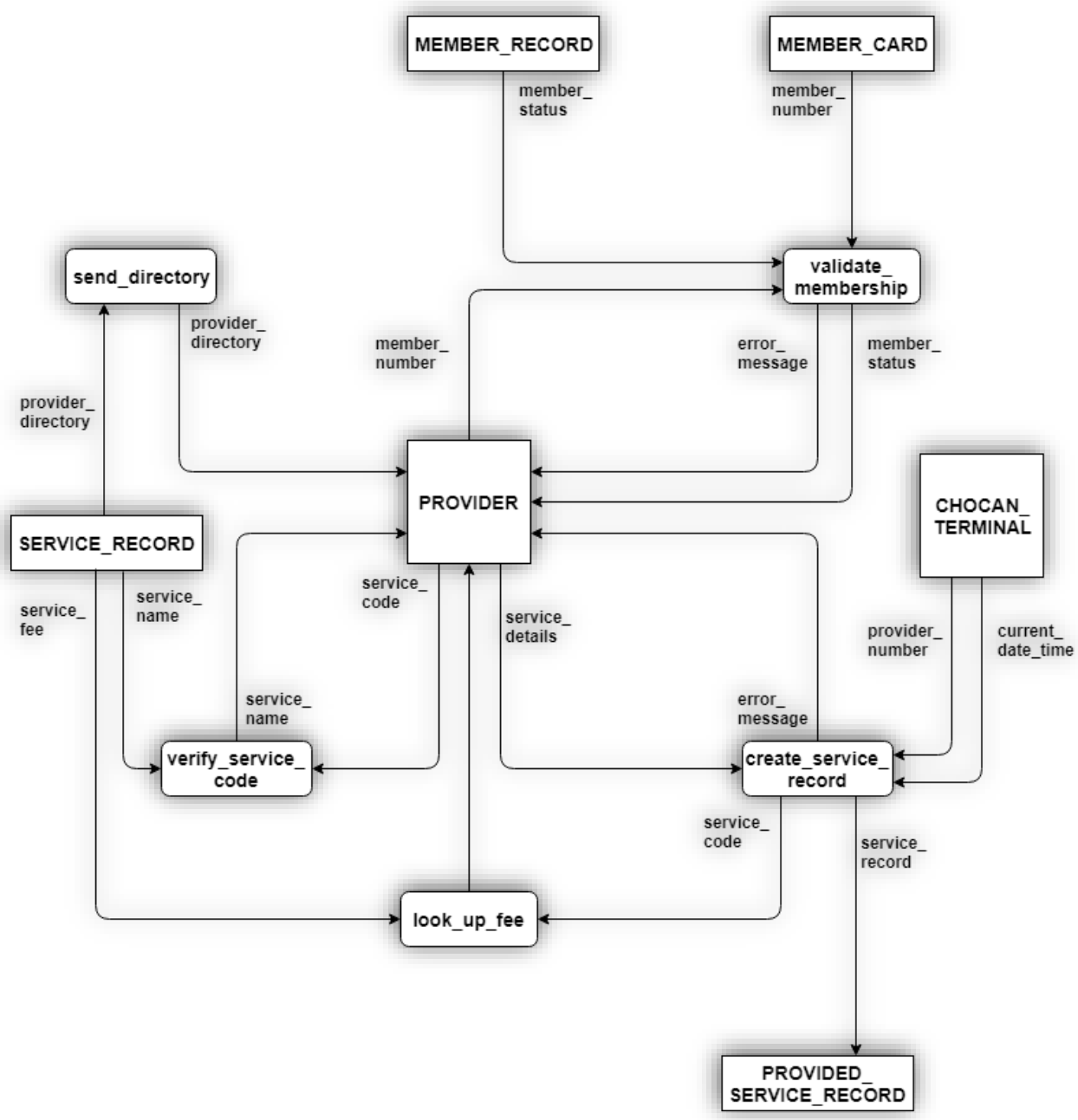


Figure 3 – Logical data flow diagram for provider interaction with the Chocoholic Anonymous product.



STEP 2: Sections to Computerize and How

The information given in Appendix A indicates that the entire process of the ChocAn product should be computerized. All provider interactions with the ChocAn terminal require computer interaction and a central database has to be installed at the ChocAn Data Center due to the large number of records that have to be stored.

It is recommended that most of the files be handled with batch processing since a large number of files have to be processed, and constricted controls are required to keep track of the services provided. This will enable reliable information processing and accurate tracking of financial information.

STEP 3: Details of the Data Flows

Table 1 – Data dictionary for the Chocoholic Anonymous product

Name of Data Element	Description	Narrative
ACME_ACCOUNTING_ SYSTEM	Accounting System	A system used by Acme Accounting Services to track member status and process EFT payments.
add_member	Procedure: Input: member_details Output: member_details	It receives the necessary member_details as input from a CHOCAN OPERATOR and outputs the member_details to add a member to ChocAn system.
add_provider	Procedure: Input: provider_details Output: provider_details	It receives the necessary provider_details as input from a CHOCAN OPERATOR and outputs the provider_details to add a member to a ChocAn system.
calculate_fees	Procedure: Input: service_code	It takes a service code as input and searches for the service_fee associated with the service_code in the SERVICE_RECORD. The service_code and the

	service_fee Output: fee_total	service_fee are then used as inputs to calculate the fee_total which is the output. The output is used to generate the create_provider_report, create_summary_report, and the calculate_sum_of_all_fees.
calculate_sum_of_all_ fees	Procedure: Input: fee_total service_fee Output: sum_fee_total	It receives the fee_total as input from the calculate_fees procedure and outputs the sum of the values as sum_fee_total, which is used to generate the create_summary_report procedure.
CHOCAN_TERMINAL	Electronic terminal	A type of computer allocated to all providers of ChocAn services for the purpose of communicating with the ChocAn data center.
CHOCAN_OPERATOR	Employee	An employee of ChocAn that is authorized to add new members, delete members and update member records.
create_eft_record	Procedure: Input: eft_record Output: eft_record	It takes the necessary eft_details as input and writes the given information as output to the EFT_RECORD.
create_member_report	Procedure: Input: member_record service_record Output: member_report	It takes the member_record and service_record for a particular week for every MEMBER as inputs and outputs a member_report. This report is generated every Friday at midnight, or anytime a MANAGER requests a report. The output is sent as an email attachment to the manager or to a member.
create_provider_report	Procedure: Input: provider_details fee_total service_details Output: provider_report eft_record	It takes the provider_details, fee_total, and service_details from the report as inputs and generates the provider_report as output which is sent as an email attachment to each provider and the manager upon request. Every Friday at midnight, the report is generated by the system. It also outputs the eft_record to process payments.
create_service_record	Procedure: Input: current_date_time provider_number	It takes the provider_number, current_date_time, and service_details as the inputs at the CHOCAN_TERMINAL and outputs the service_record to the

	service_details Output: service_record service_number	PROVIDED_SERVICE_RECORD. The service_number is also outputted to the CHOCAN_TERMINAL as the look_up_fee procedure if the service_code is correct, else it outputs an error_message.
create_summary_report	Procedure: Input: weekly_details fee_total sum_fee_total Output: summary_report	It takes the weekly_details, fee_total, and sum_fee_total as inputs and outputs the summary_report. The report is generated every Friday at midnight and sent as an email attachment to the MANAGER for accounts payable, or the report is generated upon request by the MANAGER.
current_date_time	Date of service provided	Date in the format MM-DD-YYYY
eft_details	Record comprising fields: fee_total …	It contains all details pertaining to the eft_record. IT also contains the necessary information needed to transfer funds to a particular PROVIDER. The rest of the field are yet to be determined by ChocAn.
EFT_RECORD	Bills record storage	It stores information regarding payment details necessary for transferring electronic funds to the account of the PROVIDERs.
error_message	String	An error message that is displayed to the CHOCAN_TERMINAL
fee_total	Float	Total number of services in a particular week in the format $99,999.99
look_up_fee	Procedure: Input: service_code Output: service_fee	It takes a service_code as input from the create_service_record procedure and outputs the corresponding service_fee as output to a PROVIDER.
MANAGER	Employee	A manager a ChocAn.
MEMBER	Customer	A person who registered for ChocAn services.
MEMBER_CARD	Plastic card with magnetic strip	A card given to ChocAn members with their information encoded on its magnetic strip.
member_details	Record of member fields	A record of field inputted in order to modify a member_record.
member_name	25-character string	Contains the name of a member
member_number	9-digit integer	Contains a unique number associated with a member.
MEMBER_RECORD	Record comprising fields: member_name member_number	It stores the information of each registered member of ChocAn. It is used to create a member report.

	street_address city state zip	
member_report	Procedure: Record comprising fields: member_name member_number street_address city state zip date_provide provider_name service_name	A report of member ChocAn activities generated on Friday at midnight and sent to a member as an email attachment.
member_status	String	Contains the current membership status of a ChocAn member.
PROVIDED_SERVICE_RECORD	Service provided record storage	Stores the information about all services provided to a member.
PROVIDER	Employee	A ChocAn employee that provides treatments or consultations to ChocAn members.
PROVIDER_RECORD	Provider record storage	Stores information about each ChocAn provider.
provider_details	Record comprising fields: provider_name provider_number street_address city state zip	Fields needed to generate a provider_report.
provider_directory	Document	An alphabetically orders list of service names and their corresponding service codes and fees.
provider_name	25-character string	Contains the name of a ChocAn provider.
provider_report	Record comprising fields: provider_name provider_number street_address city state zip date_provided	A list of services provided by a provider to members in the order of the date it was received by the system.

	current_date_time member_name member_number service_code service_fee total_consultations fee_total	
send_directory	Procedure: Input: directory_request Output: provider_directory	It takes directory_request as input and returns the provider_directory as an output which is sent as an email attachment to a PROVIDER upon request.
service_code	6-digit integer	A unique number associated with a ChocAn service.
SERVICE_RECORD	Service record storage	Stores all the services currently provided by the PROVIDERS.
service_details	Record comprising fields: date_provided member_number service_code comments	This fields are inputted by a PROVIDER when creating a service_record.
service_fee	Float	The cost a provided service in the format $999.99
service_name	20-character String	It contains the name of a ChocAn service.
service_record	Record comprising fields: current_date_time date_provided provider_number member_number service_code comments	It contain information regarding a service provided.
sum_fee_total	Float	The sum total of a fees to be paid by ChocAn to PROVIDERS in particular week.
summary_report	Document with fields: provider_name total_consultations fee_total total_providers sum_total_consults sum_fee_total	A summary of a PROVIDER’s work related activities for a particular week.
update_member	Procedure: Input:	It receives the necessary member_details as input from an OPERATOR or the member_status from the

	member_details member_status Output: member_details	ACME_ACCOUNTING_SYSTEM, and outputs the member_details to modify a member information using the update_member_record procedure.
update_member_record	Procedure: Input: member_details Output: member_details	It receives the necessary member_details as input from an OPERATOR and outputs the member_details to add, update, or delete a member in the ChocAn system.
update_provider	Procedure: Input: provider_details Output: provider_details	It receives the necessary provider_details as input from an OPERATOR and outputs the provider_details to modify a provider information using the update_provider_record procedure.
update_provider_record	Procedure: Input: provider_details Output: provider_details	It receives the necessary provider_details as input from an OPERATOR and outputs the provider_details to add, update, or delete a member in the ChocAn system.
validate_membership	Procedure: Input: member_number Output: member_status	It takes a member_number as input by swiping a member card at a terminal or keying a member's number by a PROVIDER. It displays a member_status such as "Invalid number" or "Member suspended" as output.
verify_service_code	Procedure: Input: service_code Output: service_name	It takes service_code as input and returns the corresponding service_name as output if it exists, otherwise and error message is displayed.
weekly_details	Record comprising fields: provider_name total_consultations total_providers	Filed necessary for creating a summary report

STEP 4: Logic of the Processes

Figure 4 – Decision tree depicting the modification of member records

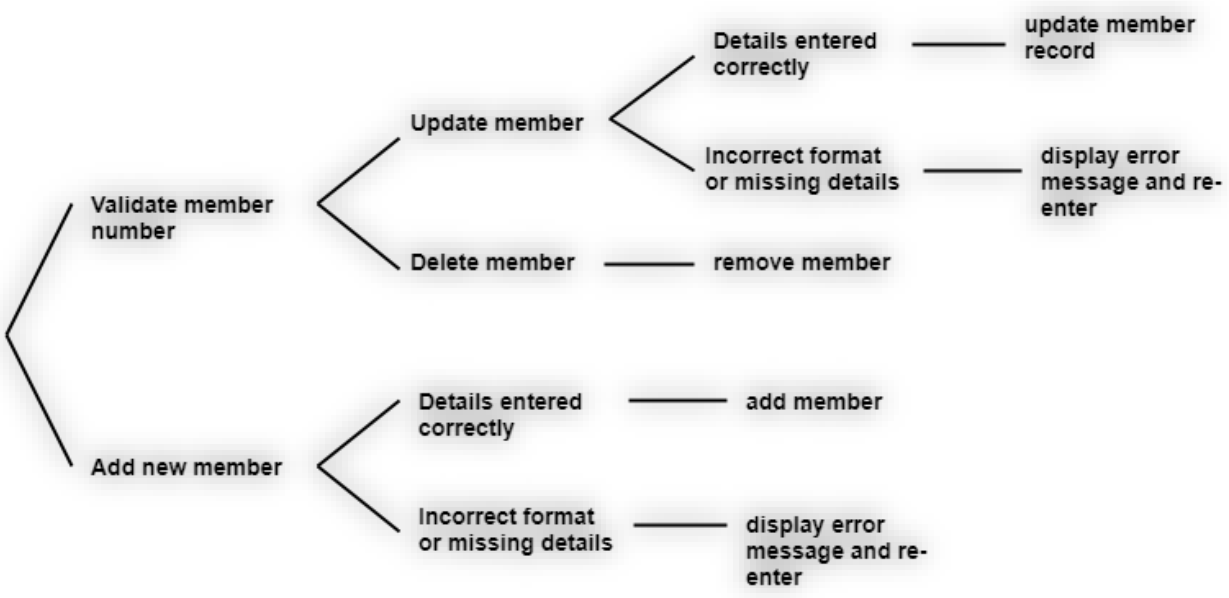


Figure 5 – Decision tree depicting the modification of provider records

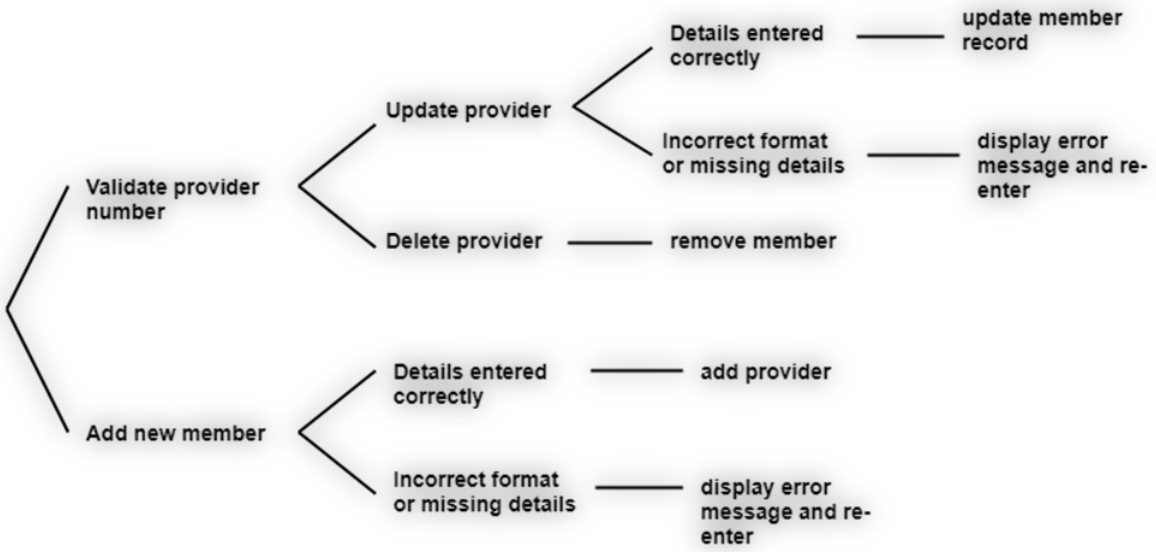


Figure 6 – Decision tree depicting service code entry by a provider

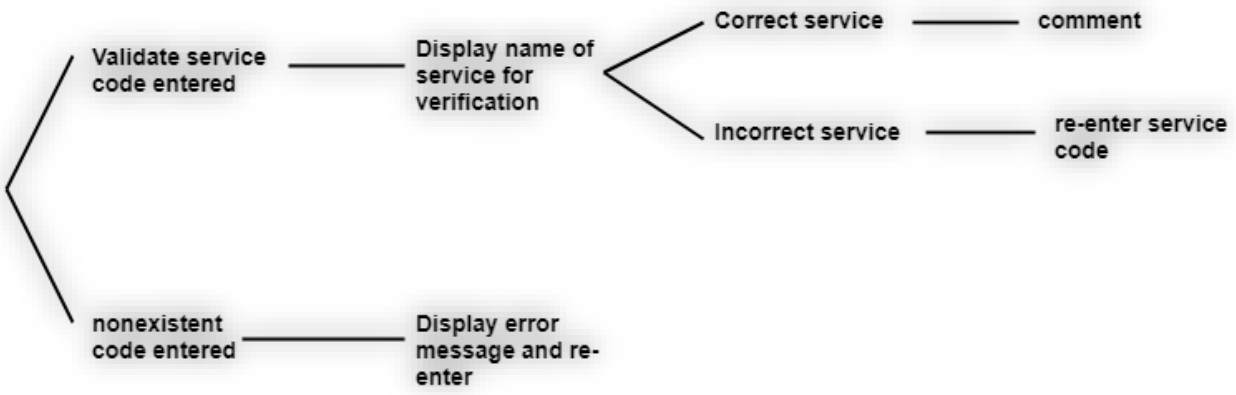


Figure 7 – Decision tree depicting ChocAn member validation messages

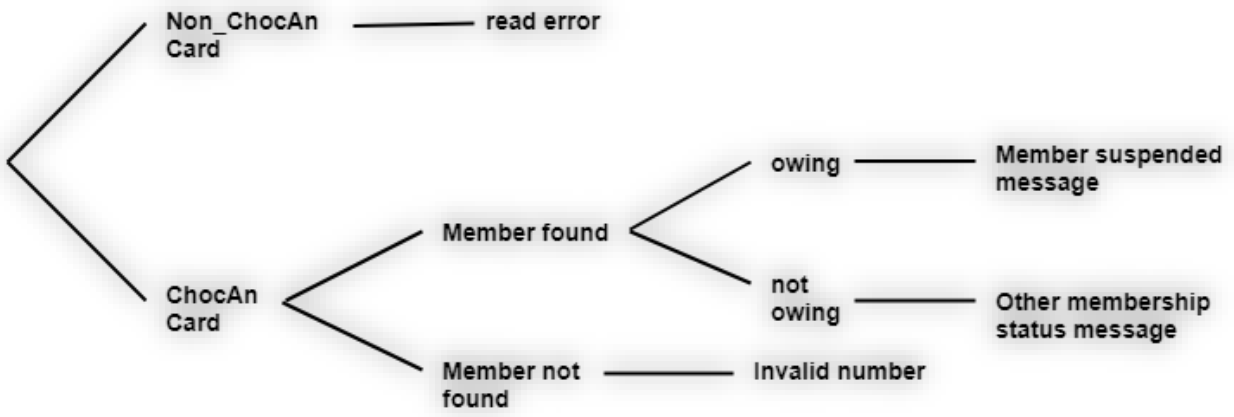
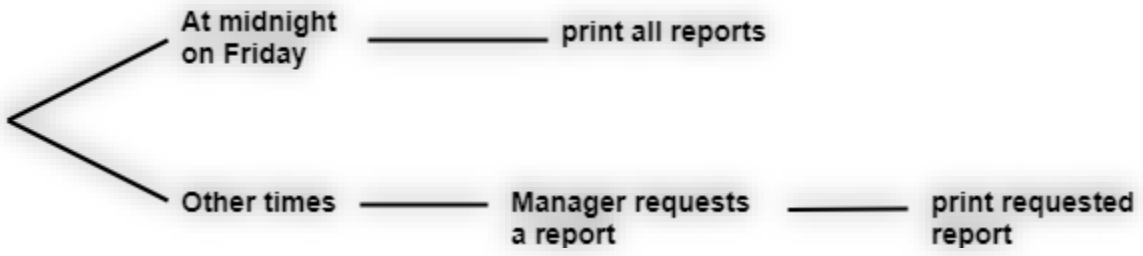


Figure 8 – Decision tree depicting report generation



STEP 5: Definition of the Data Stores

Figure 9 – The data immediate access diagram for SERVICE_RECORD

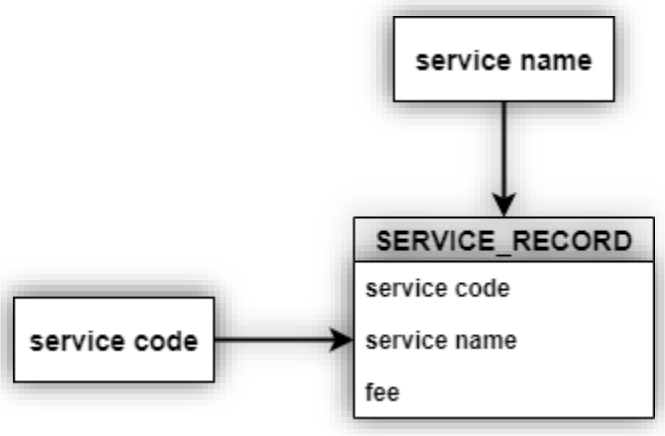


Figure 10 – The data immediate access diagram for EFT_DATA

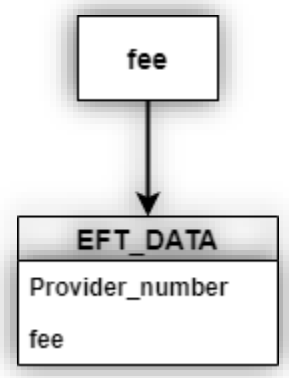


Figure 11 – The data immediate access diagram for MEMBER_CARD

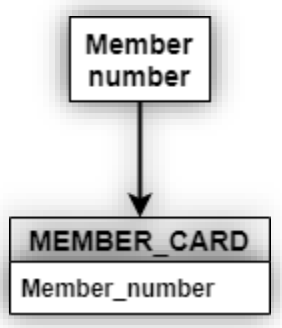


Figure 12 – The data immediate access diagram for MEMBER_RECORD

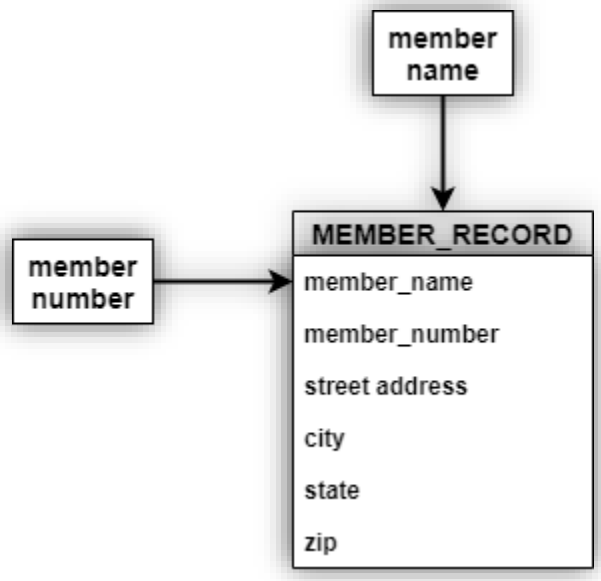


Figure 13 – The data immediate access diagram for PROVIDER_RECORD

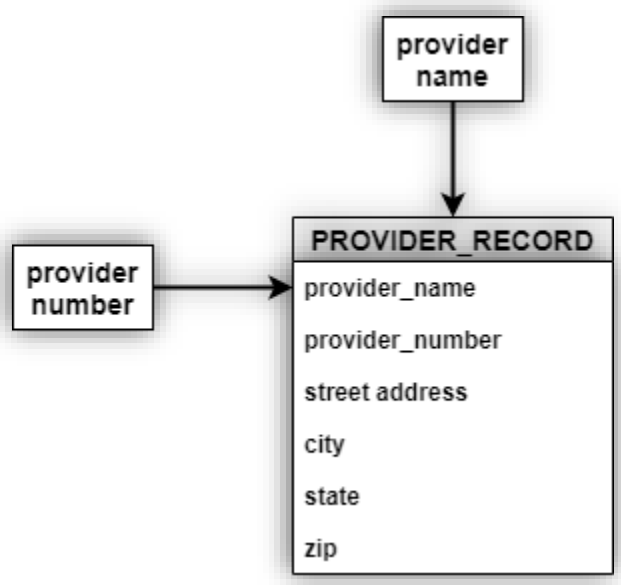
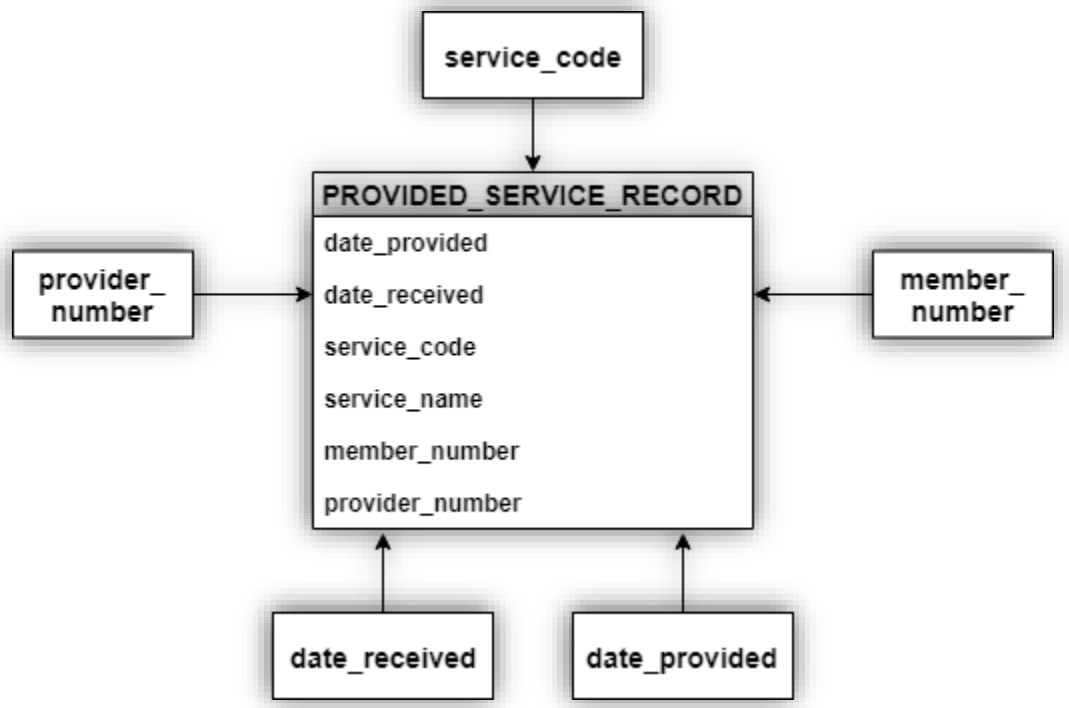


Figure 14 – The data immediate access diagram for SERVICE_CODE



STEP 6: Definition of the Physical Resources

The file names for member reports should begin with the name of the member, followed by the date the report was generated (e.g. Alex_Mason_02/02/2019). The file names for provider reports should begin with the name of the provider, followed by the date the report was generated. Depending on how often the report could be requested by a MANAGER in a day, the file names for the member and provider reports may also include the time the report was generated.

A relation database should be used with the following tables:

- MEMBER TABLE with member_number has primary key.
- PROVIDER TABLE with provider_number has primary key.
- SERVICE TABLE with service_code has primary key and service_name as a secondary index
- REPORT TABLE with the member_number, provider_number, service_code, and service_name has foreign keys, and the date_provided and date_received has secondary indexes.

STEP 7: The Input-Output Specifications

The Inputs screen include the following:

- A Login Screen for members, providers, operators, and Manager to sign into their ChocAn accounts.

- An operator Screen that allows an operator to add new members or providers, and modify a member or provider's information.
- A manager should have a similar screen to that of the operator but in addition, a manager's screen should also have options to print the member, provider, and summary reports.
- The add member/provider and update member/provider screen should have textboxes that allow an operator or manager to input information, and a button to submit it.

The output screen should including the following:

- An email attachment of a member's report containing the member's number, name, street address, city, state, zip, and the services received for the week.
- An email attachment of a provider's report containing the provider's number, name, street address, city, state, zip, services provided for the week, and the fee.
- An email attachment of a summary report containing the provider's name and number, the member's name and number, the date the service was provided, and the total fees for accounts payable.

STEP 8: Sizing

The information in Appendix A does not give enough details that would help in calculating the size of the data needed for the product. Also, if we have information about the average number of providers and members that ChocAn is expected to have in a year, we might be able to estimate the size of the reports because we know that three types of report (member, provider, summary reports) have to be printed weekly.

STEP 9: Hardware Requirements

Since we are not able to determine the sizing in step 8, the mass storage requirements cannot be computed at this time.

In terms of printing, the weekly member reports and provider reports have to be sent as an email attachment to the members and providers respectively. Only the summary report has to be printed weekly thus, not much printing has to be done and therefore will not need a high-end printer. A cheap ink-jet printer will be okay for the printing.

A fast reliable network is definitely going to important for any professional business organization. We need to have a fast and reliable communication between the ChocAn terminals and the ChocAn Data Center. The type of internet service used will depend on the network companies available in the city where ChocAn operates, and the type of connection services they offer, and their corresponding prices.

Computerized terminals are needed for providers so that they can swipe member cards or key in member numbers in order to check membership status and to record information about services provided.

A database management system is needed to store ChocAn records.

13.22. Perform the analysis workflow of the Chocoholics Anonymous product described in Appendix A.

Functional Modelling: Scenarios for the use cases in **problem 11.24**

Table 1 – Extended scenario of "Produce Report" use case.

It is Friday at midnight and the main accounting procedure is run at the ChocAn Data Center to read the week's file of services provided and print a number of reports.

1. The system reads the week's file of services provided.
2. The system checks for the members that receive services for the week.
3. The system generates a report for each member containing a list of services received during the week, sorted in order of service date.
4. The system writes a copy of the report to disk.
5. The system emails a copy of the report to the associated member.
6. The system checks for the providers that provided services for the week.
7. The system generates a report for each provider containing a list of services provided during the week and the total fees.
8. The system writes a copy of the report to disk.
9. The system emails a copy of the report to the associated provider.
10. The system writes a record consisting of the EFT data to disk.
11. A summary report is sent to the manager for accounts payable.

Possible alternatives

A. A provider did not record appropriate information for a provided service during the week.
B. The system fails to send some emails because some member/providers have the wrong email or no email on record.
C. A hardware or network failure prevents the use case from executing.

Table 2 – Another Extended scenario of "Produce Report" use case.

A ChocAn manager requests a report.

1. The manager selects the option to produce a report for a provider in a particular week.
2. The system looks up the provider information corresponding to the given provider's number.
3. The system reads a file of all services rendered by a provider in a particular week.

4. The system generates a report the provider containing a list of services provided during the week and the total fees.
5. The report is displayed on the manager's screen.

Possible alternatives

A. The manager selects the wrong report generation option.
B. The manager input the wrong provider number.
C. The system cannot find the record for the given provider number and the selected week.
D. A hardware or network failure prevents the use case from executing.

Table 3 – Extended scenario of "Add Provided Service" use case.

A ChocAn provider wants to add information about a service provided to a ChocAn member.

1. The provide swipes the member's card through the terminal or manually keys in the member's number.
2. The word "Validated" appears on the terminal screen.
3. The provider key's in the date the service was provided.
4. The provider verifies the service code.
5. The provider decides whether to add a comment or not.
6. The system checks for the service fee.
7. The provider records some information about the service on a hard copy form.

Possible alternatives

A. The member's status is actually not valid and the provider did not validate the member's card earlier before providing services.
B. The provider enters the wrong date.

Table 4 – Extended scenario of "Validate Membership" use case.

A ChocAn provider wants to check the membership status of an individual/member is valid.

1. The provide swipes the member's card through the terminal or manually keys in the member's number

2. The system uses the member's number to check the member's status.
3. The system displays the member's status on the terminal screen.

Possible alternatives

A. The card is damaged, deactivated, or is not a ChocAn member card, so the terminal cannot read the card.
B. Hardware or network failure interrupts the communication between the terminal and the ChocAn Data Center.

Table 5 – Extended scenario of "Request Provider directory" use case.

A ChocAn provider requests the software product for a provider directory.

1. The provider requests a provider directory from the system.
2. The system generates an alphabetically ordered list of service names and corresponding service codes and fees.
3. The system sends an email attachment containing the provider directory.

Possible alternatives

A. No email address is associated with the provider or the email address is wrong.
B. Hardware or network failure interrupts the communication between the terminal and the ChocAn Data Center.

Table 6 – Extended scenario of "Modify Member/ Provider Record" use case.

A ChocAn operator wants to edit the information of a member or provider.

1. The operator selects the modify member/provider option.
2. The operator decided whether to add a new member/provider, update a member/provider's record, or delete a member/provider's record.

Possible alternatives

A. The operator selects the wrong option (i.e. the operator selects "modify member" instead of "modify provider" or vice versa).
B. A hardware or network failure prevents the use case from executing.

Table 7 – Extended scenario of "Verify Service code" use case.

A ChocAn provider wants to make verify the correct service code for a provided service.

1. The provider keys in a service code corresponding to the service provided.
2. The system checks to the service name associated with the given service code.
3. The system displays the service name on the terminal.
4. The system asks the provider to verify that this is indeed the service that was provided.

Possible alternatives

A. The provider enters a non-existent service code.
B. The name associated with the service code is wrong.
C. The provider does not verify the service name.
D. Hardware or network failure interrupts the communication between the terminal and the ChocAn Data Center.

Table 8 – Extended scenario of "Lookup Service fee" use case.

A ChocAn system must look up a service fee and display it on the terminal screen.

1. The system receives a service code from a provider.
2. The system uses the provided service code to check for its associated fee.
3. The system displays the service fee on the terminal screen.

Possible alternatives

A. No associated fee exists for the inputted service code.
B. The fee displayed on the terminal screen is wrong.
C. Hardware or network failure interrupts the communication between the terminal and the ChocAn Data Center.

Table 9 – Extended scenario of "Add Record" use case.

A ChocAn operator wants to add a new member/provider record to the ChocAn database.

1. The operator selects the add member/provider record option.
2. The operator inputs the necessary member/provider information.
3. The operator submits the entry.

4. The system writes the submitted information to disk.

Possible alternatives

A. The operator enters the wrong information.
B. The operator does not enter all of the necessary information.
C. A hardware or network failure prevents the use case from executing.

Table 10 – Extended scenario of "Update Record" use case.

A ChocAn operator wants to modify an existing record in the ChocAn database.

1. The operator selects the update member/provider record option.
2. The operator inputs a member/provider's number.
3. The system searches for the record associated with the inputted member/provider record.
4. The system displays the existing member/provider's fields for editing.
5. The operator selects the field that needs editing.
6. The operator enters the new information.
7. The operator submits the edited information.
8. The system re-writes the record to disk.

Possible alternatives

A. The operator enters a wrong member/provider number.
B. The operator inputs incorrect information.
C. The operator does not input all the necessary information.
D. A hardware or network failure prevents the use case from executing.

Table 11 – Extended scenario of "Delete Record" use case.

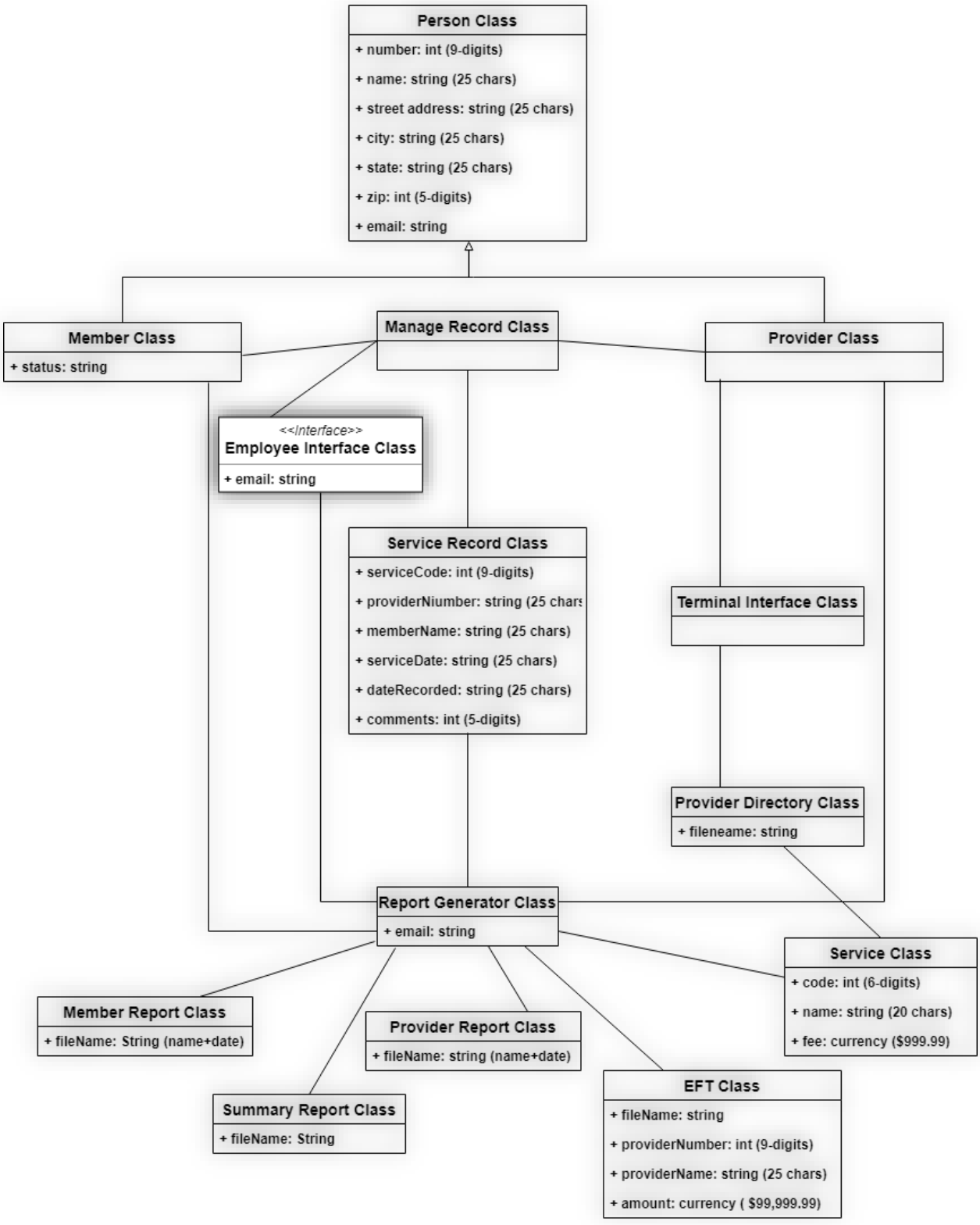
A ChocAn operator wants to delete an existing record in the ChocAn database.

1. The operator selects an update member/provider record option.
2. The operator inputs a member/provider's number.
3. The system searches for the record associated with the inputted member/provider record.
4. The system displays the existing member/provider's information.
5. The operator selects to delete to record.
6. The system removes the record from disk.

Possible alternatives

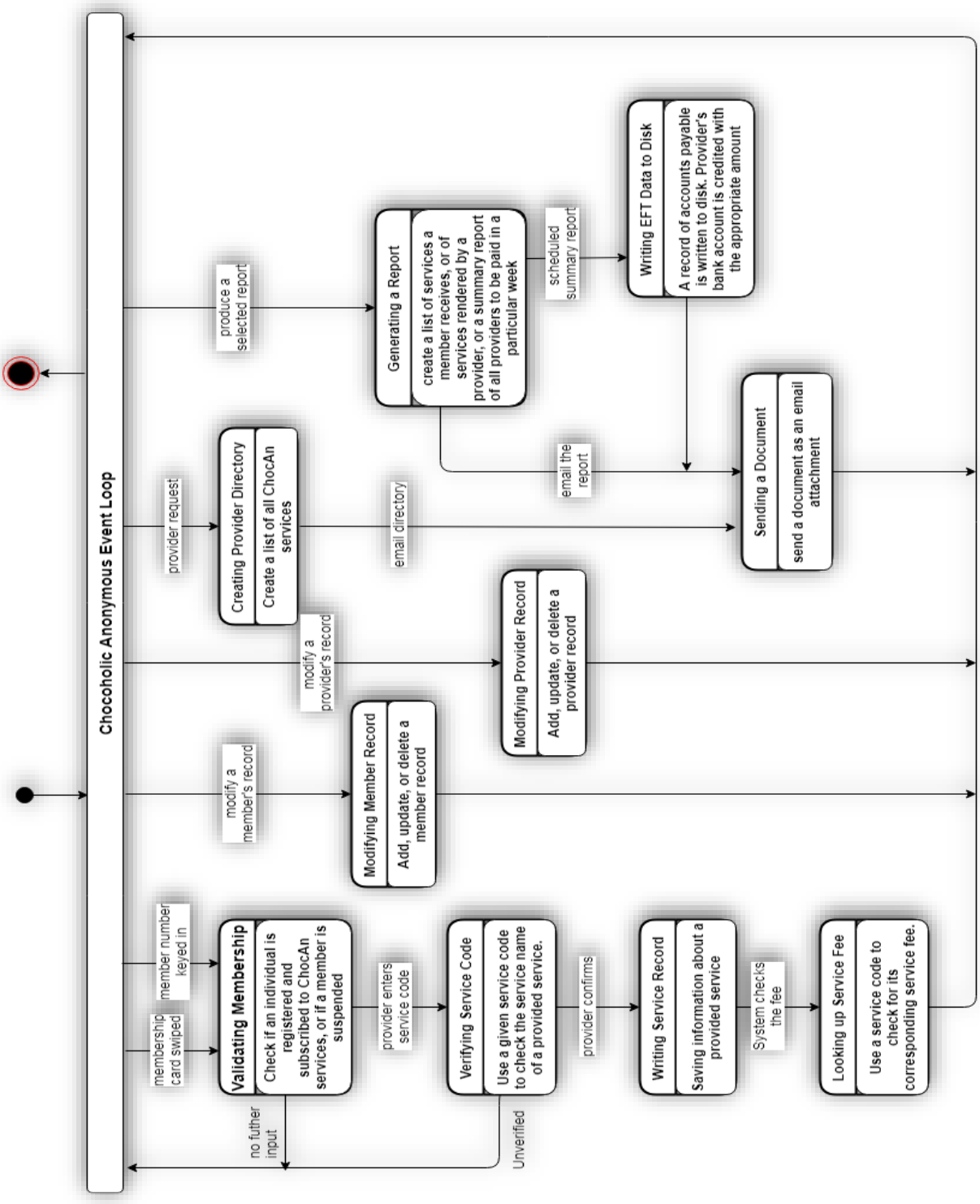
A. The operator enters a wrong member/provider number.
B. The record has already been deleted.
C. A hardware or network failure prevents the use case from executing.

Entity Class Modelling: Class Diagram



Dynamic Modelling: State chart

Figure 15 – State Chart of Chocoholic anonymous



14.16. Starting with your specification of Problem 12.20 or 13.22, design the Chocoholics Anonymous product (Appendix A). Use the object-oriented design technique.

Figure 16 – The detailed design of the modules of ChocAn

Module name	**add_member**
Module type	Function
Return type	**int**
Input arguments	**name : string, number : int, street_address : string, city : string, state : string, zip : int**
Output argument	none
Error messages	none
Files accessed	none
File changed	none
Modules called	none
Narrative	This module will create a new member record with the inputted arguments. If the module call is successful, it will return a value of 1, otherwise it will return a value of -1.

Module name	**get_member_info**
Module type	Function
Return type	**Member**
Input arguments	**number: int**
Output argument	**name : string, number : int, street_address : string, city : string, state : string, zip : int**
Error messages	**“invalid input”**
Files accessed	**member_record**
File changed	none
Modules called	none
Narrative	This module will receive a member’s number, look up the record corresponding to the number, and return a **Member** object containing all the member’s stored information.

Figure 16 – *(continued)*

Module name	**update_member**
Module type	Function
Return type	**int**
Input arguments	**name : string, number : int, street_address : string, city : string, state : string, zip : int**
Output argument	none
Error messages	none
Files accessed	**member_record**
File changed	**member_record**
Modules called	none
Narrative	This module will read a member's record and modify a member's stored information using the input arguments. If an input argument is set to "0", it will not be edited, otherwise it will be edited. If the module call is successful, it will return a value of 1, otherwise it will return a value of -1.

Module name	**delete_member**
Module type	Function
Return type	**int**
Input arguments	**number: int**
Output argument	none
Error messages	none
Files accessed	**member_record**
File changed	**member_record**
Modules called	none
Narrative	This module will receive a member's number, look up the record corresponding to the number, and delete the member record if it exists, otherwise it will return -1.

Figure 16 – *(continued)*

Module name	**verify_service_Code**
Module type	Function
Return type	**int**
Input arguments	**number: int**
Output argument	none
Error messages	none
Files accessed	**service_record**
File changed	“no record found”, “invalid input”
Modules called	none
Narrative	This module will check is the inputted service code is valid or not. If the module call is successful, it will return a value of 1, otherwise, it will return a value of -1.

Module name	**look_up_service_code**
Module type	Function
Return type	**string: string ($) + int (999.99)**
Input arguments	**number: int, string: (date)**
Output argument	none
Error messages	none
Files accessed	**service_record**
File changed	“no record found”, “invalid input”
Modules called	**verify_service_Code**
Narrative	This module will look up the fee of a Service corresponding to an inputted service code number. The **verify_service_Code** is called to check if the input is valid. If the record is found, it will return the name of the service. Otherwise, it will return an error message to indicate the record could not be found.

Figure 16 – *(continued)*

Module name	**generate_member_report**
Module type	Function
Return type	**int**
Input arguments	**number: int, date: string**
Output argument	**name : string, number : int, street_address : string, city : string, state : string, zip : int, date : string (MM-DD-YY), provider_name : string, service_name : string**
Error messages	none
Files accessed	**service_record, member_record**
File changed	none
Modules called	**service_record, get_member_info**
Narrative	This method will use the input argument to get the member's record, search through the service_record files for the inputted date range, and generates a report. It is then emailed to the associated member. If the file is created and emailed successfully, it will return a value of 1, otherwise, a value of -1.

Module name	**generate_provider_report**
Module type	Function
Return type	**int**
Input arguments	**number: int, date: string**
Output argument	**name : string, number : int, street_address : string, city : string, state : string, zip : int, date : string (MM-DD-YY), member_name : string, member_number : int, service_code : int, fee : string($) + int(999.99), t_no_consulatations: int, t_week_fee : string($) + int(99,999.99)**
Error messages	none
Files accessed	**service_record, provider_record**
File changed	none
Modules called	**service_record, get_member_info**
Narrative	This method will use the input argument to get the provider's record, search through the service_record files for the inputted date range, and generates a report. It is then emailed to the associated provider. If the file is created and emailed successfully, it will return a value of 1, otherwise a value of -1.

Figure 16 – *(continued)*

Module name	**generate_eft_record**
Module type	Function
Return type	**int**
Input arguments	**number: int, fee : : string($) + int(99,999.99)**
Output argument	none
Error messages	none
Files accessed	**service_record, provider_record**
File changed	**eft_record**
Modules called	**service_record, get_member_info**
Narrative	This method will use the input argument to get the provider's record, search through the service_record files for the inputted date range, and generate an EFT reord for accounts payable.

Module name	**generate_summary_report**
Module type	Function
Return type	**int**
Input arguments	**number: int, date: string**
Output argument	**provider_name: string, t_wk_consulatations: int, t_wk_fee : string($) + int(99,999.99), t_wk_active_providers: int, t_active_provider_consultations : int, overall_fee_total : int string($) + int**
Error messages	none
Files accessed	**service_record, provider_record**
File changed	**summary_report**
Modules called	**service_record, get_member_info**
Narrative	This method will use the input argument to get the provider's record, search through the service_record files for the inputted date range, and generates a report. It is then emailed to the manager. If the file is created and emailed successfully, it will return a value of 1, otherwise, a value of -1.

15.33. Draw up black-box test cases for the product you specified in Problem 12.20 or Problem 13.22. For each test case, state what is being tested and the expected outcome of that test case.

Figure 17 – Black Box Test Cases

Provider data:

Equivalence classes for number.

1. Character instead of digit	Error
2. Decimal instead of integer	Error
3. Negative digit	Error
4. < 9 digits	Error
5. 9 digits	Acceptable
6. > 9 digits	Error

Equivalence classes for name.

1. Any character not alphabetical	Error
2. < 1 characters	Error
3. 1 character	Acceptable
4. Between 1 and 25 characters	Acceptable
5. 25 characters	Acceptable
6. > 25 characters	Error

Equivalence classes for street_address.

1. Any character not alphabetical	Error
2. < 1 characters	Error
3. 1 character	Acceptable
4. Between 1 and 25 characters	Acceptable
5. 25 characters	Acceptable
6. > 25 characters	Error

Equivalence classes for city.

1. < 1 characters	Error
2. 1 character	Acceptable
3. Between 1 and 14 characters	Acceptable
4. 14 characters	Acceptable
5. > 14 characters	Error

Equivalence classes for state

1. Any character not alphabetical	Error
2. 2 letter code not matching state	Error
3. < 1 characters	Error

4. 1 character — Acceptable
5. 2 characters — Acceptable
6. > 2 characters — Error

Equivalence classes for zip

1. Character instead of digit — Error
2. < 5 digits — Error
3. 5 digits — Error
4. > 5 digits — Error

Equivalence classes for email

1. Not in email format — Error
2. In email format — Acceptable

Member data:

Equivalence classes for number.

1. Character instead of digit — Error
2. Decimal instead of integer — Error
3. Negative digit — Error
4. < 9 digits — Error
5. 9 digits — Acceptable
6. > 9 digits — Error

Equivalence classes for name.

1. Any character not alphabetical — Error
2. < 1 characters — Error
3. 1 character — Acceptable
4. Between 1 and 25 characters — Acceptable
5. 25 characters — Acceptable
6. > 25 characters — Error

Equivalence classes for street_address.

1. Any character not alphabetical — Error
2. < 1 characters — Error
3. 1 character — Acceptable
4. Between 1 and 25 characters — Acceptable
5. 25 characters — Acceptable
6. > 25 characters — Error

Equivalence classes for city.

1. < 1 characters — Error
2. 1 character — Acceptable
3. Between 1 and 14 characters — Acceptable
4. 14 characters — Acceptable
5. > 14 characters — Error

Equivalence classes for state

1.	Any character not alphabetical	Error
2.	2 letter code not matching state	Error
3.	< 1 characters	Error
4.	1 character	Acceptable
5.	2 characters	Acceptable
6.	> 2 characters	Error

Equivalence classes for zip

1.	Character instead of digit	Error
2.	< 5 digits	Error
3.	5 digits	Error
4.	> 5 digits	Error

Equivalence classes for email

1.	Not in email format	Error
2.	In email format	Acceptable

Equivalence classes for status

1.	Contains “Invalid number” message	Acceptable
2.	Contains “Member suspended” message	Acceptable
3.	Contains “Validated” message	Acceptable
4.	Contains any other message	Error

Service record information:

Equivalence classes for service_code

1.	Character instead of digit	Error
2.	Decimal instead of integer	Error
3.	Negative digit	Error
4.	< 6 digits	Error
5.	6 digits	Acceptable
6.	> 6 digits	Error

Equivalence classes for service_name

1.	Any character not alphabetical	Error
2.	< 1 characters	Error
3.	1 character	Acceptable
4.	Between 1 and 20 characters	Acceptable
5.	20 characters	Acceptable
6.	> 20 characters	Error

Equivalence classes for fee

1.	Character instead of digit	Error
2.	< 2 digits after decimal	Error

3. 2 digits after decimal — Acceptable
4. > 2 digits after decimal — Acceptable
5. < $ 0.00 — Error
6. Between $ 0.00 and $ 999.99 — Acceptable
7. > $ 999.99 — Error

Equivalence classes for service_date

1. Not in MM-DD-YYYY format — Error
2. Month < 01 — Error
3. Month between 01 and 12 — Acceptable
4. Month > 12 — Error
5. Day < 01 — Error
6. Day between 01 and month max — Acceptable

Equivalence classes for date_recorded

1. Not in MM-DD-YYYY HH:MM:SS format — Error
2. Before date_recorded — Error
3. After date_recorded — Acceptable
4. Month < 01 — Error
5. Month between 01 and 12 — Acceptable
6. Day < 01 — Error
7. Day between 01 and month max — Acceptable
8. Hour < 00 — Error
9. Hour between 00 and 24 — Acceptable
10. Hour > 24 — Error
11. Minute < 00 — Error
12. Minute between 00 and 59 — Acceptable
13. Minute > 59 — Error
14. Second < 00 — Error
15. Second between 00 and 59 — Acceptable
16. Second > 59 — Error

Equivalence classes for comments

1. < 1 characters — Error
2. Between 1 and 100 characters — Acceptable
3. > 100 characters — Error

EFT data:

Equivalence classes for amount

1. Character instead of digit — Error
2. < 2 digits after decimal — Error
3. 2 digits after decimal — Acceptable
4. > 2 digits after decimal — Error
5. < $ 0.00 — Error
6. Between $ 0.00 and $ 99,999.99 — Acceptable

7. > $99,999.99 Error

Equivalence classes for filename

1. Character not alphabetic Error
2. Wrong file extension Error
3. Meets file name specification Acceptable

Figure 17 – Functional Analysis Test Cases for ChocAn

The functions outlined in the specification document are used to create test cases:

1. Add a new member.
2. Add a provided service.
3. Add a provider.
4. Delete a member.
5. Delete a provider.
6. Create a provider directory.
7. Generate a member report.
8. Generate a provider report.
9. Generate a summary report.
10. Update a member's information.
11. Update a provider's information.
12. Verify a member's membership status.

In addition to direct tests, it is necessary to perform the following additional tests:

13. Attempt to add a new member that already exists on file.
14. Attempt to add a new provider that already exists on file.
15. Attempt to add a provided service already on file.
16. Attempt to create a provider directory and send it to a provider with an incorrect/non-existent email on file.
17. Attempt to create a provider report twice and check that only one EFT record is created.
18. Attempt to create each report for the current week and check it includes only information up to the current date.
19. Attempt to delete a member not on file.
20. Attempt to delete a provider not on file.
21. Attempt to update a member not on file.
22. Attempt to update a provider not on file.
23. Attempt to update a member's information twice and check that the second version is stored.
24. Attempt to update a provider's information twice and check that the second version is stored.
25. Attempt to verify a member not on file.
26. Attempt to verify a member with an "Invalid" or "suspended" status.

16.15. Suppose that the product for Chocoholics Anonymous in Appendix A has been implemented exactly as described. Now the product has to be modified to include endocrinologists as providers. In what ways will the existing product have to be changed? Would it be better to discard everything and start again from scratch? Compare your answer to the answer you gave to Problem 1.19.

In **Problem 1.19**, using my basic experience with object-oriented languages like Java, I noted that "it would not be necessary to discard everything and start again because adding a new type of provider (an endocrinologist) is not a complex issue. We can take advantage of Object-Oriented Programming by incorporating abstraction, inheritance, and polymorphism to our program. A provider can be a dietitian, internist, exercise experts, or an endocrinologist (polymorphism) by creating a "provider" superclass and using abstraction to make sure that its methods are implemented in its subclasses (dietitian, internist, exercise experts, or an endocrinologist). Then, the subclasses inherit the methods and fields of its superclass.".

However, after doing the project and case study, I maintain that it would not be necessary to discard everything and start again because adding a new type of provider is not a complex issue assuming we are using the object-oriented paradigm. We would have to determine the best way to implement the new functionality by using cost-benefit analysis to examine the needed changes. We would only need to make changes to a few simple attributes of the Provider class, and we would need to include an Endocrinologist class which would be a sub-class of the Provider superclass. This maintenance changes should be small and relatively easy to implement, so the cost-benefit analysis outcome will most likely favor not starting a new product from scratch.

If the cost-benefit analysis is used under the classical paradigm, the addition of an endocrinologist as a provider will entail that new provider types and services will need to be added to the existing project. Many of the modules of the system will need to be able to work with this new provider type, hence, the developers will have to change some the source code, which may be difficult but less expensive than starting from scratch. It will also take more time to restart the project than to just modify the project to add the needed changes.

As long as a specification document and all other necessary documentation have been drawn, the cost benefit analysis it will likely favor modify the product to starting from scratch. We will only have to edit the specification document and the code to include the needed changes.

References

Schach S. R (2011). *Object-Oriented and Classical Software Engineering, 8th ed*. New York: McGraw-Hill, 2011.

Tools

UML Diagrams: *https://www.draw.io/*