COMP 410
Software Engineering

Assignment 1
Term Project

Jason Bishop
3042012

June 29, 2015

Chapter 1
Problem 1.19

Suppose that the product of Chocoholics Anonymous of 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?

The Chocoholics Anonymous software will have to be revised in a few ways to include the addition of endocrinologists as providers. New provider numbers will have to be issued if the endocrinologists exist as separate providers and not part of an existing provider group. Additionally, new types of services that they provide will need to be added to the system, along with their corresponding names, fees, and codes. This information must also be included in an updated Provider Directory. The rest of the system will need to be able to handle these changes.

If the system were designed *exactly* as described in Appendix A, then these additions would be difficult to implement because the system is tailored to the specific criteria listed. This means the system can only handle dieticians, internists, and exercise experts. Correspondingly, the system will only be able to handle the treatment options available at implementation. This includes the names, fees, and codes of services. If the provider directory is a simple document, this may be able to be edited for changes in a word-processing (or similar) program, but these changes will not be reflected in the system.

As such, I would recommend that the company indeed discard everything and start from scratch. This way, the system can be designed so that post-delivery maintenance of this kind is easily done and can quickly meet client needs. The best way to achieve this result would be to design the system under the object-oriented paradigm. For example, the terminal at the service provider's location sends a message containing the information mentioned. The object representing the account of the provider could then be updated with whatever the message contains, and the same goes for the member object. Internally, the objects can process the information however they wish. Continuing the example, a database might store the list of services and associated information. The objects in question could simply query the database to provide the information to the users. Then to update the system with new information, simply requires an addition to the database.

Therefore, the best solution is to discard everything and redesign the software from scratch under an object-oriented paradigm. This will extend the life cycle of the product because updates to the Chocoholics Anonymous program can be implemented with relative ease to keep the software current and relevant to meet changing client and user needs. This will also reduce overall costs, because maintenance is such a large part of the total cost of software development. And finally, designing the product in this way meets the ethical standards of software engineering by acting in the best interests of the client, and the product will be meeting the highest of professional standards.

Chapter 2
Problem 2.22

Which software life-cycle model would you use for the Chocoholics Anonymous product described in Appendix A? Give reasons for your answer.

For the Chocoholics Anonymous product described, I would use the iterative-and-incremental life-cycle model. The main reason is that this model most closely represents real-world software development. A model that closely represents the real world is more likely to be accurate. Better accuracy enables the software development team to deliver a product that is more likely to be on time and on budget. This model has been proven to work, as mentioned in section 2.7 of the textbook.

There are additional reasons that this mode should be chosen to develop the Chocoholics Anonymous product. The iterative-and-incremental model has many opportunities to check that the software is correct, the robustness of the architecture is determined early, and there is always a working version of the software. These three benefits would be advantageous in pretty much any software development situation. However, the particular software development situation in this example requires these benefits in a unique manner.

The company in the example has been contracted to develop only a portion of the total system. This is the data processing software that the system will use. Therefore, this software is only a piece of the entire system and will have to fit in exactly if the entire system is to work properly. For example, if the data processing software did not integrate with the communications software correctly, the resulting data may be mishandled or inaccurate. This will lead to errors that may have negative effects on customers and health providers. Therefore, if the data processing software is always in a working state, the latest version can be tested with the other developing parts of the system. Also, if the architecture's robustness is checked early, then any necessary changes can be made early in the cycle, and will allow other developing systems to adapt to these changes early. Finally, the multiple opportunities to check for correctness ensure that the interacting systems will be working with an accurate model of the software, not one that contains several errors.

The iterative-and-incremental life-cycle model is a very good choice for the development of the data processing software for the Chocoholics Anonymous system. There are several positive aspects of this cycle that work well with this particular situation. One of these aspects is the more realistic view of software development. The other aspects provide needed benefits because the product being developed is only a piece of the system that will need to integrate with other software products developed elsewhere. The benefits of the iterative-and-incremental model help the software integrate into the system as a whole much better. Therefore, this seems to be the best choice for selecting a developmental life-cycle model.

Chapter 3
Problem 3.15

What differences would you expect to find if the Chocoholics Anonymous product of Appendix A were developed by an organization at CMM level 1, as opposed to an organization at level 5?

If the Chocoholics Anonymous produce were developed by an organization at CMM level 1, the management taking care of the product would be practically non-existent. This level, called the "initial level," is characterized by it's ad-hoc approach to software development. Therefore the data processing section of this product that is being developed is likely to be plagued with problems. For instance, the data types for incoming data may be arbitrarily chosen, without consulting the company doing the communications. When the time comes to integrate the product, the data types supplied may be different than expected. The organization will have to go back and fix the code. Additionally, the product will probably be coded such that it is not encapsulated very well, causing the programmers to have to sift through the code to find the faults caused by the type difference. This in turn is likely to delay the product significantly.

Also, because the development is likely to take a non-trivial amount of time, there will be staffing changes during the development life-cycle. These kinds of changes at the initial level will also cause problems for the development of this software. For example, one programmer may not document his or her code very well. If this person is fired, quits, or leaves the organization for some other reason, the replacement programmer will have trouble coming into the situation. Specifically, they will likely have difficulty understanding the code, and may have to start again from scratch. This again is likely to cause a delay in delivery of the product, and the additional programming time will also add to the development cost. These kinds of time and cost delays are quite common at CMM level 1.

There would be a vast difference with the same product being developed by an organization at CMM level 5. This level is referred to as the "optimizing level," and is characterized by continuous improvement of the software development process. As such the time and cost delays present at level 1 are much less likely. In the data typing example above, a level 5 organization is likely to be using object-oriented programming techniques. This allows for encapsulation and re-use of code. Therefore the development process can be more efficient because the company can re-use processes from earlier projects to help develop the Chocoholics Anonymous project, and if the code needs to be changed, it is often much easier. However, the coding changes present in the example are much less likely to occur, because the organization will have planned and prepared for these types of circumstances.

The other example, regarding staffing changes, is also unlikely to occur within an organization at the optimizing level. All code should be meticulously documented because the organization will have much stricter quality and process controls. That way, any new person coming into the project can easily familiarize themselves with the necessary information. This goal of constant, consistent improvement means that the level 5 organization is much more likely to deliver code that is on time, on budget, and more capable of expansion and change. In other words, the product is more likely to meet the client's needs.

Chapter 4
Problem 4.15

What type of team organization would be appropriate for developing the Chocoholics Anonymous product described in Appendix A?

Mod chief or hierarchy?

From the many team organizational models presented in the textbook, it seems that most of them are not very feasible for the Chocoholics Anonymous project. The extreme ends of the team organizational spectrum are occupied by the democratic team and the chief programmer team. However, there are some severe limitations to both these approaches. The democratic team can not be imposed from an external source. It must develop as a natural process within the team. This does not make it very reliable in a business sense, because the business can not count on it being developed, and if a developer leaves the team, there is no guarantee that the team will be able to reform around a replacement. The chief programmer is limited by the demands of the roles required for the team. The chief programmer must be highly skilled in both programming and management, which are rare finds even on their own. Additionally, the backup programmer will need to have the same skills, but be willing to work for less money. Finally the programming secretary's job is centred mainly around paperwork, which is not ideal for most developers.

Yet even the modern team organizations that occupy more central places on the spectrum, still have very limiting faults. For example, the teams for agile processes are centred around pair programming. However, experiments have shown that pair programming takes more effort, and that success is highly dependent on both the programming expertise of team members as well as the complexity of the developed system and tasks involved. This again seems unreliable in a business sense, because the business cannot guarantee that the software development proposals it receives will meet these criteria. Additionally, if the business only accepts those contracts which they feel meet the criteria, the organization ends up limiting itself in its options. Open-source programming teams are impractical for pretty much any development business. Open-source teams are centred around unpaid volunteers, and rarely have any type of requirements, analysis or other organizational structures. Essentially, no guarantees for success can be made with this type of development, and therefore, no client will wish to take such risks in developing any type of software this way.

There are more team organization methods that have found success, but only in rare cases. The synchronize and stabilize method has been successful for the Microsoft corporation, but no where else of significance. This model has also been criticized because it does not guarantee any type of success. Microsoft has hired many talented individuals within their organization, and the results may be because of talent as opposed to team methods. Also, some argue that the success of Microsoft products has come from marketing, rather than this type of organizational structure. There is also the case of the modified chief programmer team that has had limited success. The greatest success of this organizational model came way back in 1971 with the New York Times clipping file project. No chief programming team has had the same success since that time, even though many projects have been developed with this method.

The one type of team left is the modern hierarchical programming team. It's weaknesses lie in the fact that the separation of technical and non-technical areas of a development project can lead to confusion and problems unless responsibilities are clearly delineated. However, if these are laid out before the start of the project, the weaknesses can be avoided and an organization can then take advantage of its strengths. These strengths are a combination of the democratic and chief programmer approaches, without the drawbacks. The modern hierarchical team has a team leader and a team manager for the technical and non-technical aspects of the project respectively. This eliminates the need for a chief programmer which, as mentioned above, can be quite difficult to find. This team structure can also employ a decentralized communication channel when needed to take advantage of the collaborational aspects of the democratic team. Finally, one the best aspects of the hierarchical team is that it can scale. Small projects can be organized into small hierarchies, while large projects can add additional layers to the team structure. For these benefits and lack of crippling weaknesses, the modern hierarchical programming team seems the best approach for developing the Chocoholics Anonymous project.

Chapter 5
Problem 5.17

What types of CASE tools would be appropriate for developing the Chocoholics Anonymous product described in Appendix A?

The Chocoholics Anonymous project described is not so large that it needs a massive amount of personnel and resources to accomplish. Yet that does not mean that it can not benefit from the use of a number of CASE tools to aid with the development of the project. There are many CASE tools available to provide benefits at every stage of the software development life-cycle, and the developing organization in this case should take advantage of these tools.

Some of the best tools that the organization should use come within an effective programming workbench. This includes a structure editor with online interface checking, an operating system front-end, a source-level debugger and online documentation. A structure editor "understands" the implementation language, which can help with syntax and format of the code. An interface checker tracks the names of variables and methods so that the types and number of variables used will consistently match and easily be found. The operating system front end allows operating system commands to be issued from within the editor itself. A source level debugger can automatically trace errors produced in the code. Overall, these tools can help the programmers on the team develop the project faster, with fewer faults, better usability, and easier maintenance.

Other case tools that can be an advantage in the Chocoholics anonymous project are online documentation, Web browsers, spreadsheets and word processors. These tools are main used to keep documentation for the project accurate, up-to-date, and easily accessible. Developers, including programmers and maintenance staff, will be able to quickly change and update the documentation. This will help speed the development of the product. This will also allow end-users to have easy access to documentation they may need when using the software when installed and used in the real-world environment.

A consistency checker used in conjunction with a data dictionary can also be valuable tools for development of this project. The data dictionary tracks all data within items within the project including descriptions, names, types, and locations of variables and procedures. The consistency checker ensures that every data item in the specification document is reflected in the design, and vice-versa. This ensures that the product is developed according to specifications and will meet the client's needs accurately. Also, any missing information is easily accounted for and can quickly be developed. This tracking allows developers to build a more accurate product.

In general, there are several CASE tools that could potentially benefit the development of the Chocoholics Anonymous product. Developing and maintaining documentation is an important aspect of development, and many of these tools can help with that process. Additionally, the implementation and maintenance of the product is aided with several tools available in a programming workbench. These allow developers to create and maintain code easily, leading to a better development process. In fact, there are probably many more types of tools beyond this discussion that will aid the development of this project. The organization must assess the time and financial costs of implementing each of these CASE tools into the development of the product.

Chapter 6
Problem 6.17

Explain how you would test the utility, reliability, robustness, performance, and correctness of the Chocoholics Anonymous product in Appendix A.

The Chocoholics Anonymous product needs to be rigorously tested to ensure it meets the client's needs. Testing should be done at all stages of the software development life cycle, but as with all software products, testing is most intensive during the implementation workflow. At this time, the product is tested for utility, reliability, robustness, performance and correctness. These test must be performed before the product is delivered to the client. Therefore, these tests are necessary for the Chocoholics Anonymous product.

Testing the utility of a product means that we must meet user needs when using the correct product under the correct conditions, laid out by the product specifications. The first part of a utility test is to test how easy the product is to use. This can be tested by having the client come in and work with the user interface. Close attention should be paid to the user interface and how easily the client can enter and access information. The next part of the utility test is to determine if the product performs useful functions. This can be tested by ensuring that the reports generated are well-formatted and accurate. The final part of a utility test is to determine if the product is cost effective compared to competing products on the market. Since this is custom software for Chocoholics Anonymous, this is harder to test. However, off the shelf products with similar functionality can be examined for their price point and how well they fulfil the specifications of this project. This information can be compared to the projected budget and previous utility tests to determine if the product is viable.

A reliability test measures the frequency and criticality of product failures. Metrics such as the mean time between failures and the mean time to repair can be used to help determine reliability. This information can be tracked throughout the testing project. As tests, inspections, and walkthroughs are performed the number and type of faults detected should be recorded. The time taken to address the faults should also be kept. This information can be compared to other projects to determine if the Chocoholics Anonymous project is performing worse than average. If so, this is a warning sign for management to begin taking corrective action.

The robustness test relies on many factors: The range of operating conditions, the possibility of unacceptable results with valid input, and the acceptability of effects when the product is given invalid inputs. This can be tested using well designed test cases using both correct and incorrect input parameters. Any faults found in the software should be quickly addressed so the product remains robust. The product should also be designed under the object oriented paradigm, so that it receives and produces messages. This encapsulation and information hiding will help make the product robust as well, and this can be tested using well-designed test cases.

Testing of performance involves determining the extent to which the product meets the constraints of response time and space requirements. This information should be determined from the client specifications. Throughout development this should constantly be targeted and tested. Space requirements are relatively easy to test. The product should be compiled and linked, then the size of all needed files for the system should be calculated to determine if the product meets the size requirements. The response time requirements can be tested with well designed test cases. These test cases should test the limits of the system as much as possible, including using incorrect information. Response times should be measured and recorded to compare with the specifications laid out.

Correctness checks make sure that the product satisfies its output specifications, independent of the use of computing resources, when operating under permitted conditions. This can be done with correctness proofs. These proofs should be developed in tandem with the code development to make the process easier. Developers should spend time working through the mathematical proof of the system. However, it should be determined beforehand if the extra cost of correctness checking is necessary, using a cost-benefit analysis.

Chapter 7
Problem 7.23

Suppose that the Chocoholics Anonymous product of Appendix A was developed using the classical paradigm. Give examples of modules of functional cohesion that you would expect to find. Now suppose that the product was developed using the object-oriented paradigm. Give examples of classes that you would expect to find.

If the Chocoholics Anonymous product were to be developed using the classical paradigm, there would be many modules displaying functional cohesion. To handle the member verification process with the ChocoAn terminal modules such as **`verify_member_number`** and **`check_member_status`** will check that the member is in the system and return the status of the member to the terminal. The next step using the terminal is for the user to key in the service code a module such as **`look_up_service_code`** can be used to return the information associated with the code or an error message. After the user enters information about the provided service the module **`write_record_to_disk`** can be used to store the information. Then the software has to determine the fee for that particular record using the **`look_up_fee`** module. This will return the fee to the user and store it with the associated record.

There are also various reports that must be generated by the software. For example, a provider can request a list of all services via email. This can be done with a module such as **`send_provider_list`**. Another report is the weekly member's report, automatically generated with the **`send_member_report`** module. Each provider who has billed the company also receives a report, generated with the module **`send_provider_report`**. Finally, a financial report is generated using the module **`send_financial_summary_report.`** All of these reports can be requested at any other time, with corresponding request modules. Additionally, each module should read the database for the information and be able to format the report properly.

Additional modules need to be added to be able to keep the system up-to-date with member and provider information. Modules like **`add_member`**, **`delete_member`**, and **`update_member`**, as well as their counterparts for providers, will be used. Lastly, to track the financial information of each transaction, the module **`write_eft_to_disk`** can be used to store the information for the modules developed by the other company to handle the necessary electronic funds transfers.

Alternatively, if the Chocoholics Anonymous product were to be developed using the object-oriented paradigm, a fewer number of classes would be needed to develop the product. One such class would be the **`member`** class, to store member information and perform all member related tasks. This would be somewhat similar to the **`provider`** class, which stores data and handles provider tasks. A class hierarchy of reports can be used with an initial **`report`** class, that can be sub-classed into the member/provider/financial summary reports using dynamic binding and polymorphism to handle the implementation details. With these small number of classes, almost all of the modules present in the classical development can be incorporated into informationaly cohesive modules.

Chapter 8
Problem 8.24

Suppose that the Chocoholics Anonymous product of Appendix A is developed using the classical paradigm. What parts of the product could be reused in future products?

First of all if the aim of the organization developing the Chocoholics Anonymous product is to promote reuse, I would recommend that they develop under the object-oriented paradigm, rather than the classical paradigm. Reuse is much less likely when using the classical paradigm. However, there is still reuse that can be achieved while developing under this paradigm. This reuse depends on what future products that the organization will be developing.

For example, the application framework of the product could be reused when developing a similar future product. The framework has already been laid out to be able to validate membership, send and receive information from terminals, and generate reports. These general functions can be reused in another software product. The organization would simply need to plug in the specific functionality of the new product. The reason that the individual models can not be reused is because the data for the modules would also need to be reused, as modules can only achieve functional cohesion under the classical paradigm.

This reuse of framework might happen if the organization were developing a membership program for a casino. Staff at each gaming table can swipe the user membership card to validate membership status. The terminal would send information about the game the user is playing and the amount of time and money spent. Finally, the casino can use the information of these play habits by aggregating it into a report. This will let them see which games are most popular or profitable. Individual members could be assigned rewards based on the amount of time and money spent in the casino. These functions can be plugged into the basic framework laid out above.

Some design patterns can also be reused in this scenario. With the Chocoholics Anonymous product, member and provider information is stored, updated, and accessed. There would be specific patterns that need to be designed to accomplish these functions. For example, an iterator or database cursor can be used to access and traverse the stored records. This same type of structure can be implemented on the casino membership data. The difference would lie in the specific implementation of the pattern with the casino's storage architecture.

Chapter 9
Problem 9.13

Consider the Chocoholics Anonymous project described in Appendix A. Why is it not possible to estimate the cost and duration purely on the basis of the information in Appendix A?

The information in Appendix A describes a basic layout for the functionality of the Chocoholics Anonymous product. However, the information presented is clearly not enough to make an estimate regarding the cost and duration for developing this project. The information provided does not provide specifics to estimate these factors. For example, there is no mention of the programming language or life-cycle model to be used for development, there are no metrics such as KDSI estimates for this type of product, and no mention of previous experience that the development company has with developing this type of software. For these reasons and more, it is impossible to give an accurate cost or duration estimate.

There are many approaches to estimation of these factors. Using expert judgement by analogy, expert opinions within the company are consolidated based on previous work experience. As mentioned above, there is no mention of previous experience provided, for the entire organization or for individuals within it. Therefore, this estimation technique cannot be used. The bottom-up approach attempts to break the product into smaller components and estimated the cost and duration of each of these components. Again as previously mentioned, the details necessary for this process. The implementation language is not specified, nor is the use of a specific life-cycle model. Additionally, the development hardware, availability of CASE tools, staff salaries, and much more. All of these inputs would be necessary for this estimation technique and many others.

Other techniques of estimation are also inapplicable. When considering intermediate COCOMO or COCOMO II, these techniques require a lot of detail for estimates that, as mentioned, are not provided. Also, they require knowledge of the company's metrics for previous work in software development to be able to calculate the estimates, which again is not provided in the basic layout of the product. A technique to provide metrics like FFP is also not feasible to use. FFP is a metric based on calculation of function points in the program, which are in turn based on the files, flows, and processes of the product. Necessary in these calculations is also the efficiency of the development process within the organization. Once again, we are lacking this information.

Overall, there is a distinct lack of organizational information, implementation information, and metric information. These factors are necessary for many models to use when providing cost and duration estimates for a software development project. There is no other information provided other than what is available in Appendix A. As such, it is not possible to produce an accurate time or duration estimate for the Chocoholics Anonymous product.

Chapter 11
Problem 11.24

Perform the requirements workflow for the Chocoholics Anonymous project in Appendix A.

To properly elicit the requirements for the Chocoholics Anonymous project, we would first need to understand the application domain. In this instance the domain is addiction treatment. People join organizations such as Chocoholics Anonymous to help them deal with addiction issues to substances and behaviours. Providing these services is not free, yet many organizations do not charge for their services, instead relying on donations or government support. Services provided directly by governments are also often free.

However, there are organizations, such as Chocoholics Anonymous, which do charge for the services provided to members. These services often include, counselling and treatments with professionals, to aid the member in overcoming their addiction. Some organizations can also be involved in research and education as well. The professional providers of these services are usually paid. This is sometimes covered by insurance or government assistance, but can also be covered by membership fees or dues from the organization itself. The organization's work involved providing access to these services, often including scheduling, payment and follow-up.

Chocoholics Anonymous has been set up specifically to help people addicted to chocolate. There is a monthly fee to be paid by members. This fee provides unlimited access to professional treatment and counselling with many types of experts. Members who have not paid the fee are denied access until their account is up to date. When joining the organization, members are provided with a membership card and number that allows them to access these services. Service providers are secured through a currently unknown process, but provide their services to members in return for payment from the Chocoholics Anonymous organization.

After gaining an understanding of the domain, key terms have been placed into the glossary seen in Figure 1.

The next step of the requirements workflow is to develop an initial business model. The development organization would interview the the staff, members, and providers of Chocoholics Anonymous to gain an understanding of it's operation. These will be the future users of the product being developed. Obviously, this is not possible to do in this situation, so let us assume that the result of any interviews, questionnaires and other requirements elicitation procedures results in the information found in Appendix A. This will allow the development of use cases for the project, presented below in Figure 13. The descriptions for each use case are presented in Figures 2 – 12.

Figure 1 – The glossary for the Chocoholics Anonymous project.

Term	Definition
Acme Accounting Services:	a third-party organization responsible for financial procedures
ChocoAn:	an abbreviation of *Chocoholics Anonymous*
ChocoAn Data Centre:	the location for the storage of all ChocoAn records, and future site of the target software product
ChocoAn terminal:	a device to read membership cards, send and receive data, similar to a credit card device, and located with a provider
Consultation:	A meeting for discuss and evaluation of a member's case and/or treatment
Dietician:	an expert in nutritional health care
EFT:	an acronym for *electronic funds transfer*
Exercise expert:	an expert in many types of personal training
Interactive mode:	the functionality of the target software product that allows for addition, update, and deletion of both member and provider records
Internist:	an expert in non-surgical diagnosis and treatment of disease
Member:	a individual paying a monthly fee to join ChocoAn
Member card:	a plastic card given to members, embossed with a name and membership number, incorporating a magnetic strip
Member number:	a 9-digit number corresponding to a registered member
Operator:	an individual working for ChocoAn, allowed to perform basic functions in the target software product.
Provider:	a health care professional who provides services to members, either a dietician, internist, or exercise expert
Provider Directory:	an alphabetically ordered list of service names and corresponding service codes and fees
Provider number:	a 9-digit number corresponding to a provider, either an individual or organization
Service:	a treatment or consultation provided by a provider
Service code:	a 6-digit number corresponding to a provided service provided
Suspended:	the status of a member's account when membership fees have not been paid for at least one month
Treatment:	the application and management of health services

Figure 2 – The description of the *Validate Membership* use case of the Chocoholics Anonymous project.

Brief Description:

The *Validate Membership* use case allows a provider to check the status of a member through the ChocoAn terminal.

Step-by-Step Description:

1. The provider swipes the member's card through the terminal.
2. The terminal contacts the ChocoAn Data Centre with the member number.
3. The system returns the status corresponding to the member number.
4. The terminal displays the status on the terminal screen.

Figure 3 – The description of the *Add Provided Service* use case of the Chocoholics Anonymous project.

Brief Description:

The *Add Provided Service* use case enables a provider to update the system with the information regarding a service provided to a member.

Step-by-Step Description:

1. Using the terminal, the provider keys the date the service was provided in the format: MM-DD-YYYY.
2. The provider looks up the appropriate service code and keys it into the terminal.
3. The *Verify Service Code* use case is invoked
4. The provider verifies the code or returns to step 2 to input a new code.
5. The provider can enter additional comments or leave the field blank.
6. The system writes a record that includes the following information:
   Current date and time (MM-DD-YYYY HH:MM:SS)
   Date service was provided (MM-DD-YYYY)
   Provider number (9 digits)
   Member number (9 digits)
   Service code (6 digits)
   Comments
7. The *Look Up Service Fee* use case is invoked.
8. The provider records the information on a paper form for later verification purposes.

Figure 4 – The description of the *Verify Service Code* use case of the Chocoholics Anonymous project.

Brief Description:

The *Add Provided Service* use case enables the *Verify Service Code* use case to verify that the service code entered is correct.

Step-by-Step Description:

1. The name of the service provided is looked up using the service code.
2. A message is sent to the terminal containing:
   a) the name of the service if the provided code is stored in the system.
   b) an error message if the provided code is not stored in the system.
3. The terminal displays the message (up to 20 characters) for the user to verify.

Figure 5 – The description of the *Look Up Service Fee* use case of the Chocoholics Anonymous project.

Brief Description:

The *Add Provided Service* use case enables the *Look Up Service Fee* use case to display the fee for a particular service.

Step-by-Step Description:

1. The service fee is looked up using the service code.
2. The fee is sent to the terminal.
3. The terminal displays the fee.

Figure 6 – The description of the *Request Provider Directory* use case of the Chocoholics Anonymous project.

Brief Description:

The *Request Provider Directory* use case sends a provider a Provider Directory

Step-by-Step Description:

1. The provider sends a request to the system for a Provider Directory
2. The system prepares an alphabetically ordered list of all services in the system with corresponding service codes and fees.
3. The list is sent to the provider's email as an attachment.

Figure 7 – The description of the *Modify Provider Information* use case of the Chocoholics Anonymous project.

Brief Description:

The *Modify Provider Information* use case modifies the provider information stored in the system.

Step-by-Step Description:

1. The operator modifies the provider information system in one of the following ways:
   a) a new provider joins Chocoholics Anonymous and the *Add Record* use case is invoked.
   b) a provider requires changes to information stored in the system and the *Update Record* use case is invoked.
   c) a provider resigns from Chocoholics Anonymous and the *Delete Record* use case is invoked.

Figure 8 – The description of the *Modify Member Information* use case of the Chocoholics Anonymous project.

Brief Description:

The *Modify Member Information* use case modifies the provider information stored in the system.

Step-by-Step Description:

1. The system modifies the stored member records when one of the following occurs:
   a) The operator modifies the member information system in one of the following ways:
      - a new member joins Chocoholics Anonymous and the *Add Record* use case is invoked.
      - a member requires changes to information stored in the system and the *Update Record* use case is invoked.
      - a member resigns from Chocoholics Anonymous and the *Delete Record* use case is invoked.

   b) Every day at 9:00pm, the Acme Accounting Services system modifies the user records stored in the system and the *Update Record* use case is invoked.

Figure 9 – The description of the *Add Record* use case of the Chocoholics Anonymous project.

Brief Description:

The *Modify Provider Information* or *Modify Member Information* use case enables the *Add Record* use case to add a new record to the system.

Step-by-Step Description:

1. The system uses input from the operator to add a new record to the system. For each record:
    a) if the record is for a provider, the following information is stored in the system:

Provider name (25 characters)
Provider number (9 digits)
Provider street address (25 characters)
Provider city (14 characters)
Provider state (2 letters)
Provider ZIP code (5 digits)

    b) if the record is for a provider, the following information is stored in the system:

Member name (25 characters)
Member number (9 digits)
Member street address (25 characters)
Member city (14 characters)
Member state (2 letters)
Member ZIP code (5 digits)

Figure 10 – The description of the *Update Record* use case of the Chocoholics Anonymous project.

Brief Description:

The *Modify Provider Information* or *Modify Member Information* use case enables the *Update Record* use case to add a new record to the system.

Step-by-Step Description:

1. The system uses input from the operator to modify a currently existing record within the system. For each record:
    a) if the record is for a provider, the system updates the provider's record with the new information.
    b) if the record is for a member, the system updates the member's record with the new information.

Figure 11 – The description of the *Delete Record* use case of the Chocoholics Anonymous project.

Brief Description:

The *Modify Provider Information* or *Modify Member Information* use case enables the *Delete Record* use case to add a new record to the system.

Step-by-Step Description:

1. The system uses input from the operator to delete a currently existing record within the system. For each record:
    a) if the record is for a provider, the system removes the provider's record from the system.
    a) if the record is for a member, the system removes the member's record from the system.

Figure 12 – The description of the *Produce a Report* use case of the Chocoholics Anonymous project.

Brief Description:

The *Produce a Report* use case prints reports and/or sends them to the appropriate managers, providers, or members.

Step-by-Step Description:

1. The system reads the stored records when one of the following occurs:
    - a) A ChocoAn manager requests a report.
    - b) Every week, at midnight on Friday, reports are requested for the following:
        - each member who consulted a ChocoAn provider during the week.
        - each provider who has billed ChocoAn that week.
        - a financial summary.
2. The system creates one or more of the following reports:
    - a) a member report with the following information:

Member name (25 characters)
Member number (9 digits)
Member street address (25 characters)
Member city (14 characters)
Member state (2 letters)
Member ZIP code (5 digits)
For each service provided, the following details are included:
Date of service (MM-DD-YYYY)
Provider name (25 characters)
Service name (20 characters)

    b) a provider report with the following information:

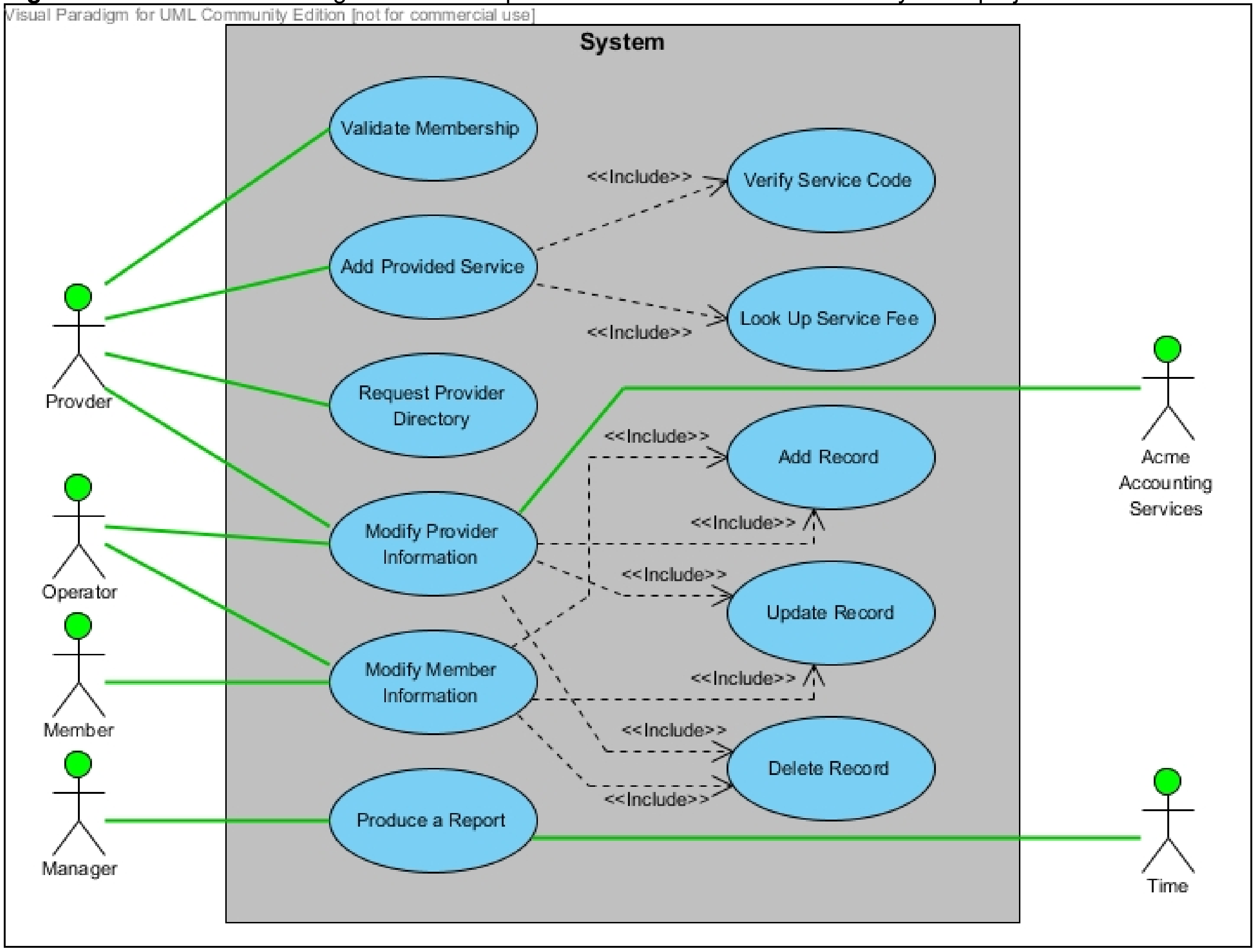
Provider name (25 characters)
Provider number
Provider street address (25 characters)
Provider city (14 characters)
Provider state (2 letters)
Provider ZIP code (5 digits)
For each service provided, the following details are required:
Date of service (MM-DD-YYYY)
Date and time data were received by the system (MM-DD-YYYY HH:MM:SS)
Member name (25 characters)
Member number (9 digits)
Service code (6 digits)
Fee to be paid (up to $999.99)
Total number of consultations with members (3 digits)
Total fee for week (up to $99,999.99)

    c) a financial summary report with the following information:

Every provider to be paid that week with the following details:
The number of consultations each provider had that week
The provider's fee to be paid that week
The total number of providers who provided services that week
The total number of consultations by all providers for that week
The overall fee total for that week

3. If it is Friday at midnight, the system writes an EFT record to disk.
4. The system prints the report.
5. If it is Friday at midnight, the reports are emailed as attachments in the following manner:
    - a) a member report to each member consulting a ChocoAn provider during the week.
    - b) a provider report to each provider who has billed ChocoAn that week
    - c) a financial summary is to the accounts payable manager.

Figure 13 – The use case diagram for the requirements of the Chocoholics Anonymous project.



The next step would be to interview the clients in an attempt to extract further details. This will allow the client a chance to update or add any new information that the developer may not have included. This information would then be used to refine the initial requirements and the use case diagrams. Then, these new requirements would again be reviewed, creating an iteration process that continues until both the developer and the client feel that all needs have been met. Again, this is impossible to do in this case, so the requirements will be left as they stand.