Comp 410
Term Project
Jason Bell 3078931
May 21, 2015

1.19 (Term Project) Suppose that the product for 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?

It would not be necessary to discard the product and recreate it.

First, based on the description of the product, no changes to the terminals used by the providers will be needed. The terminals simply display the data and prompts sent to them by the product. So existing terminals can continue to be used.

To add endocrinologists, a new service code for that service needs to be added to the Provider Directory, which would need to be redistributed to the providers.

The product's database will need to have entries added corresponding to the new endocrinologist service code, along with a name and cost to display on the provider's terminal for confirmation.

At this point, any uses of the endocrinologist service will show up in the reports like any other service.

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

I'm not sure that Appendix A gives enough information to be able to choose a software life-cycle model: it depends on the number of team members involved and their skills. There's also no mention of the client being involved in development beyond the acceptance test. Finally, there's no mention of most-delivery maintenance being part of the contract. So, here's some hypothetical scenarios.

I'd first suggest that this product is relatively simple. Based on this, I'd also suggest that the development team is small. Let's suppose 5-10 people. This is just my guess from experience programming as a research assistant. The exact number isn't important, but this is not a product that will require 100 people. I'll now consider each model.

The open source model is clearly out. This is proprietary software, not open source. It makes no sense to create the product through a paid contract and make it freely available at the same time.

The spiral model is suited for large in-house projects. This project is neither large nor in-house.

The synchronize and stabilize model has not seen success outside of Microsoft's unique culture.

I suspect that the course considers the waterfall model to “never” be appropriate, but I'd argue that it's situational. The waterfall model may be acceptable if the client's requirements do not change and the development team is not trained in agile processes, but are still highly competetent.

Code and fix is suitable for trivial projects with unclear requirements. That doesn't fit here as while this project is small, I wouldn't go so far as to call it trivial. It's requirements are also fairly defined.

Rapid prototyping may not be suitable as it requires that the client use each prototype and suggest changes. First, we don't know if the client it going to be involved aside from the final acceptance test. Second, the requirements are fairly defined.

I'll consider the evolution-tree model and iteration/incrementation models together since they are so similar. First, a major consideration with these models is client involvement. The project description does not indicate whether this will be the case other than a final acceptance test. If this is the case, this may not be an ideal model. However, the requirements are fairly defined, so the client may not need to be actively involved in the process. If the development team has the proper training, using the Unified Process would be an excellent way to further refine requirements.

So, I have two answers to this question. Choosing between them requires information that I don't possess. If the client is uninvolved until the acceptance test and the team is untrained in agile processes, the waterfall model may be acceptable: the project is small and it's requirements are reasonably clear. If the team is trained in agile processes, extreme programming with the Unified Process would be ideal. A small team will already be working closely together: may as well take it a step further and work in the same room in pairs. Even though extreme programming requires active involvement of the client, this project is simple enough and it's requirements clear enough that I'd argue that extreme programming could still work: especially if serious efforts are made to refine the requirements using interviews and questionnaires.

3.15 (Term Project) 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?

A CMM level 1 organization may produce a quality product on time and under budget if my hypothetical team of 5-10 people and their manager are high skilled and work well together. That is, a small team of super-star programmers may be able to brute force a product like this. However, even if this is the case CMM level 1 suggests no sound practices: everything is ad hoc. As such, documentation and future planning may be non-existent. This means that while the product may pass the acceptance test now, future maintenance and changes will be more expensive than they should be.

Naturally, at CMM level 1 if the team and/or manager are not highly skilled, the product will likely take too long, be over budget, be poor quality, or all of the above.

At CMM level 5 there are rigorous processes that have gained acceptance within the culture of the team. Everything is documented, crises are predicted and prevented, testing is rigorous throughout, and metrics are of statistical quality. The product should be expected to be on time, at or under budget, of high quality, with proper design and documentation to support future maintenance and changes.

4.15 (Term Project) What type of team organization would be appropriate for developing the Chocoholics Anonymous product described in appendix A?

As with the previous questions, this depends upon the nature of the team members.

My hypothetical team is small. If they've worked together for some time and work well together, a democratic team may work. However, the text does suggest that managers and experienced programmers may find it hard to accept a democratic team. These are cultural issues that would have to be considered depending on the personalities in the team.

The team and project are small enough that a hierarchy of leaders and managers is unnecessary. If a democratic team is not workable given the personalities of the team, a chief programmer and backup would be needed to manage this team. So, hypothetically, one chief programmer with one backup managing a team of 8 developers.

If the team is untrained in extreme programming, the team members would have to be assigned tasks by the chief programmer, and some would have to be assigned to testing.

If the team is trained in extreme programming, the team members could work in pairs, one testing and one coding, and sharing experience. They would also have the freedom to choose tasks from the list of pending tasks. This would also lessen the managerial responsibilities of the chief programmer, so he can spend more time working on architectures and critical code.

Clearly, the ideal is my hypothetical team of 5-10 developers working democratically and in pairs in the extreme programming model, assuming that they have the required training and expertise.

5.17 (Term Project) What types of CASE tools would be appropriate for developing the Chocoholics Anonymous product described in Appendix A?

Short answer: most of them.

Longer answer:

First, I'm not going to bother discussing ubiquitous tools such as email, web browsers, word processors, or spread sheets. The use of these tools in any modern business environment should be a given.

Data dictionary with a consistency checker: ensure that there is as source of valid data to be used by prototypes for testing and for the simulated acceptance test.

A report generator could be used to generate the provider directory and the reports. That is, a designer should be able to define their layout in a visual fashion, and the report generator could produce the code needed to generate it. This would assist making changes to the directory and reports later on: just make the changes to the visual design, and re-execute the generator.

A screen generator could be used in the same manner to create and modify the interface of the terminals used by the providers (I'm unclear from the appendix whether another company is designing just the terminals, and we're designing what's displayed on the screen, or if that company is handling both).

A requirements tracker could be useful, but may be overkill. For a project this small, and a small team working in one room, I have to wonder if a dedicated white board or requirements document would be sufficient. However, since there are free solutions available, such as bugzilla, there's no reason to not use one. A requirements tracker would be an excellent way of having a single repository of requirements and their current status. Using this software tool would help synchronize updates to a requirement and keep history, like a source repository. For example, if someone updates the status of a requirement with incorrect information, finding out who is responsible would be simple due to login info, as would reverting the status to the previous one.

Online documentation should be a given, even if it's in the form of a wiki. Wikis are free and simple to setup, track changes, and can host discussions on topics. I see no reason for the team to not have a wiki setup as a documentation repository. As things change it would be simple to update the documentation, while keeping the history, and the team can discuss the content in the wiki.

The question of structure editors and pretty printers should also be a given. In fact, these seem like dated terms to me: any IDE that doesn't support syntax validation and highlighting is a terrible one. In a modern context, given how freely available excellent IDEs are (Visual Studio Express, Eclipse, Netbeans, etc), not using one seems inexcusable. Even if one is using a custom language, creating a language extension for these IDEs is a natural step after creating the language compiler. The fact that the text also talks about the editor activating the compiler, linker, and debugger also surprises me, since these should also be a given. In my experience as a programer, it's usually an all or nothing scenario: either you're writing code in text files and building with a makefile, or using a fully featured IDE.

With a team of 5-10 people, version and source control is clearly a must. The code written by them must be synchronized to avoid overwrites and collisions. Versioning must be done so that when a test build is created, the appropriate source code is used. It would also assist with creating builds that use code from previous versions, if desired.

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

The requirements are clear enough that I see no excuse to not do test driven development, especially if efforts are made to refine those requirements with interviews and questionnaires (my experience is that test driven development is nigh impossible if you don't have clear requirements). If the team is using the extreme programming model, one developer from each pair can work on the tests, while the other writes the artifact. Each of the metrics listed in the question are testable or measurable.

To test utility, create tests that feed valid input into the artifact or whole product and check that the output matches the requirements in terms of correctness, ease of use, and cost effectiveness (for example, does the product use an expensive piece of technology when a cheaper one is available?). Ideally, a client should be on site to help determine that these conditions are met, but the appendix is unclear as to whether this is the case here.

Reliability is is how often or severely the product fails. Measuring this requires rigorous tracking of faults as they happen, and documentation of what was required to recover from the fault. If the organization is at CMM level 1, this will be challenging, as rigorous processes are not in place. At

CMM level 5, measuring reliability should be an automatic part of the process. Measuring reliability of the product or it's components help highlight areas in need of improvement. As the text says, if the product fails infrequently, but that failure is difficult to recover from, the product is still unreliable.

Robustness should be simple to test. Test the product's ability to operate under different conditions by creating a range of simulated conditions (say, with a virtual machine) under which to run the tests. Examples of such a range are different operating systems and hardware. One can test the product's ability to respond correctly to incorrect data by purposely feeding it incorrect data. The product passes the test if it correctly prompts the user with an error message rather than failing. One challenge of this kind of testing is that the developers have to try to think of every possible kind of invalid data. I've experienced this myself: I thought a product was robust thanks to my own tests, then someone else breaks it by doing something I didn't think of (a reason why ideally one shouldn't test one's own code....).

Performance would be a very important factor to test in this project, and could use the same sort of simulated environments as the robustness tests. First, the product operates from a central location and communicates with terminals at remote locations. Thus, the product must still operate at acceptable performance even if network conditions happen to be poor. This could involve buffering transactions for retransmission and ensuring that only data that absolutely needs to be sent is sent. Testing performance under poor network conditions can be done in a simulated environment: the environment could be configured to “lose” or delay packets between the product and the terminals. One simulate many other conditions such as other operating systems or limited memory or hard drive space.

My interpretation of correctness testing essentially brings the previous categories of tests together. Does the product operate correctly given valid input and the required environment? It seems to me that if it passes the previous categories of tests, it will pass the correctness test. For example, the robustness and performance testing may indicate that the product operates correctly on lesser hardware than the required hardware, but as long as it runs on the expected hardware, it will pass the correctness test. Conversely, if the product fails the performance and robustness test on the expected hardware, it certainly won't pass the correctness test. In that sense I see the correctness testing as confirmation. It also seems like an ideal opportunity to have the client test the product: run a simulated environment with the expected operating system and hardware, and have the client use the product and observe the tests.

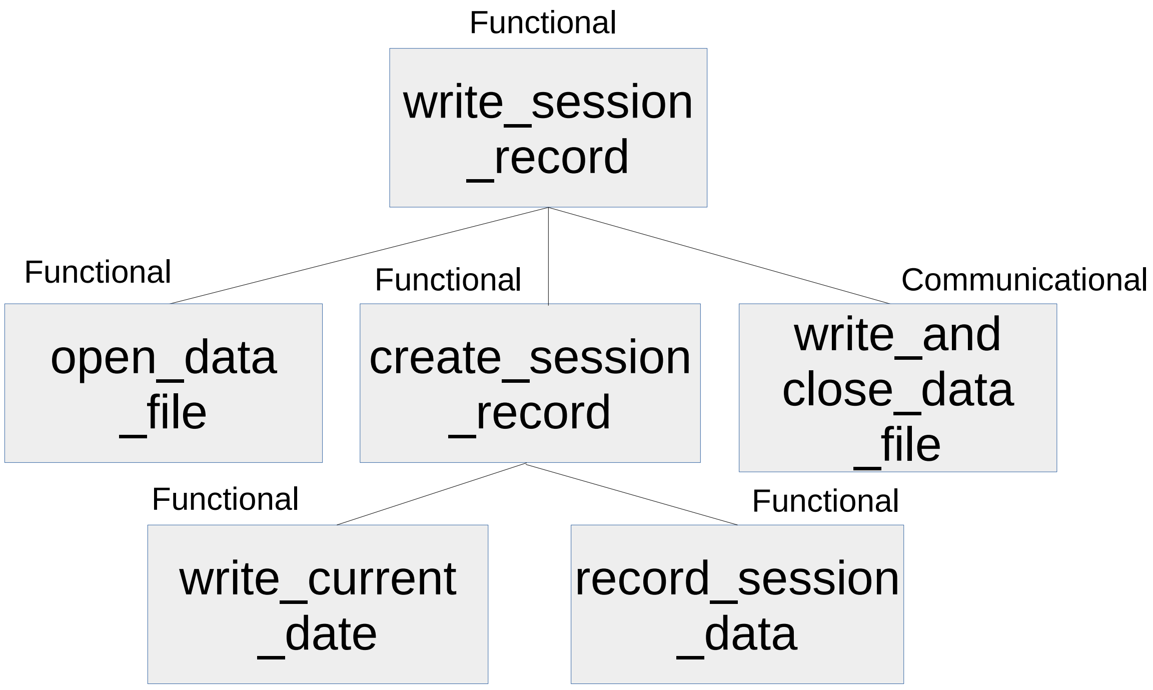
7.23 (Term Project) Suppose that the Chocoholics Anonymous product of Appendix A was developed using the classical paradigm. Give examples of modules of functional cohesion that you would expect to find. Now suppose that the product was developed using the object-oriented paradigm. Give examples of classes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

In answering this question I'll be analyzing the modules that record a session.

Recording Sessions

I will refer to these as "sessions": instances of a provided service. Note that this module doesn't care about validation of the data: that should have been performed prior to invoking this module.

Classical Paradigm:



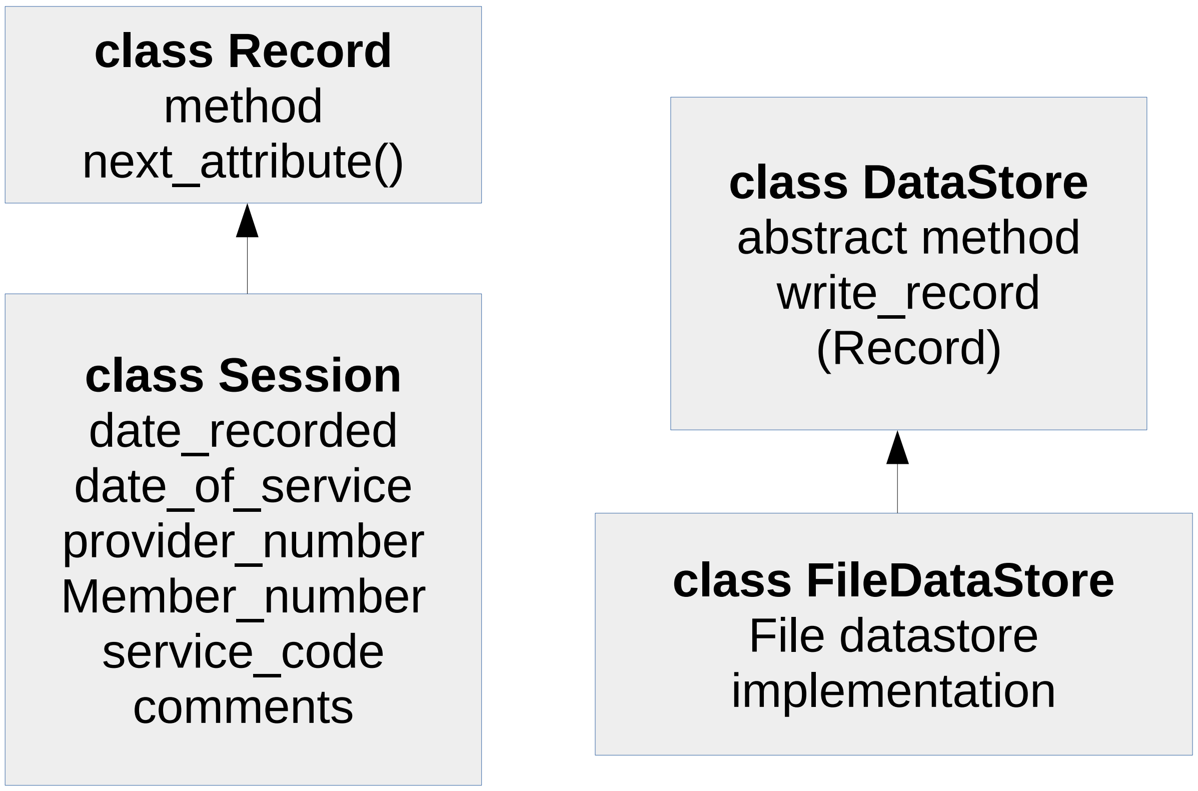
The module write_session_record is functional because it has one operation and goal. It is not informational, because that would require that it perform a number of operations, each with it's own entry point, on one data structure. In this regard, write_session_record could itself be one of the components of an informational module, along side a read_session_record module.

For the sake of brevity: all other modules except for write_and_close_data_file are not informational for the same reason: they each perform one operation with one goal.

The module write_current_date simply writes the current date at the moment to the record. The module record_session_data writes the session data to the record: the date given by the provider, the provider number, member number, service code, and comments. The modules write_current_date and write_session_data are functionally dependent upon create_session_record, because we need a record in which to record their data. However, they are not functionally dependent on each other. You don't need to first write the current date to write the session data, and vice-verse.

The module write_and_close_data_file is communicational because has multiple operations as per the steps required by the product, and on the same data (the data file). They could be split into separate write_data and close_data_file modules, dependent on write_session_record, each with functional cohesion. I leave it as is as an example of a communicational module.

Object Orientated Paradigm:



This is a more data-orientated view of this portion of the product. The previous classical model did not describe the session data at all, only a sequence of operations. The previous model also considered creation of the record to be a part of the process of writing the record to the data store: in this one the Session exists independently and is passed to write_session_record. Session is also derived from a base Record class that has a method allowing iteration through the attributes defined by classes that inherit from it.

DataStore is also abstract, such that multiple implementations writing to different media can be created. This one here writes to a file, as to the requirements in the appendix, but others that write to a cloud, or graph databases, or mySQL database. DataStore's abstract write_record method accepts Records, so that any implementation of Record can be written to the store, not just Sessions. write_record would use Record.next_attribute to walk through the attributes.

8.25 (Term Project) Suppose that the Chocoholics Anonymous product of Appendix A was developed using the classical paradigm. What parts of the product could be reused in future products? Not suppose that the product was developed using the object orientated paradigm. What parts of the product could be reused in future products?

This is a very difficult and open-ended question: in either paradigm, poorly designed code will not be re-usable, and vice-verse. However, in an ideal world, parts created with the object-oriented paradigm will be more usable overall. Again though, code doesn't need to be object orientated to be re-usable.

Using the example from the previous question, the classical module write_session_record would not be re-usable unless it's responsibilities were very similar in the new product (ie, the data store and format of session data hasn't changed). If these have changed, or the required sequence of operations of the product have changed, this code will have to be discarded and rewritten.

The object orientated code in the previous example is highly reusable: worst case scenario is that there's a new class derived from Record describing the new data format, and we need a new implementation of DataStore if we're no longer using files to store data.

9.13 Why is it not possible to estimate the cost and duration purely on the basis of the information in Appendix A?

There simply is not enough information to make accurate estimates. At this point we only have some broad requirements. We need to work through the analysis workflow to obtain more technical details from which an estimate can be made. Estimating duration and cost can require technical metrics such as lines of code, operations, function points, and numbers of files, flows, and processes. These values give an idea of the complexity of the product, and can be used in algorithms such as COCOMO II to determine cost. However, we just don't know enough to give values to those metrics.

At this point we also can't consider the human factor without these technical details: we don't know if we have people with required skills (because we don't know what skills are required!), or how many we need.

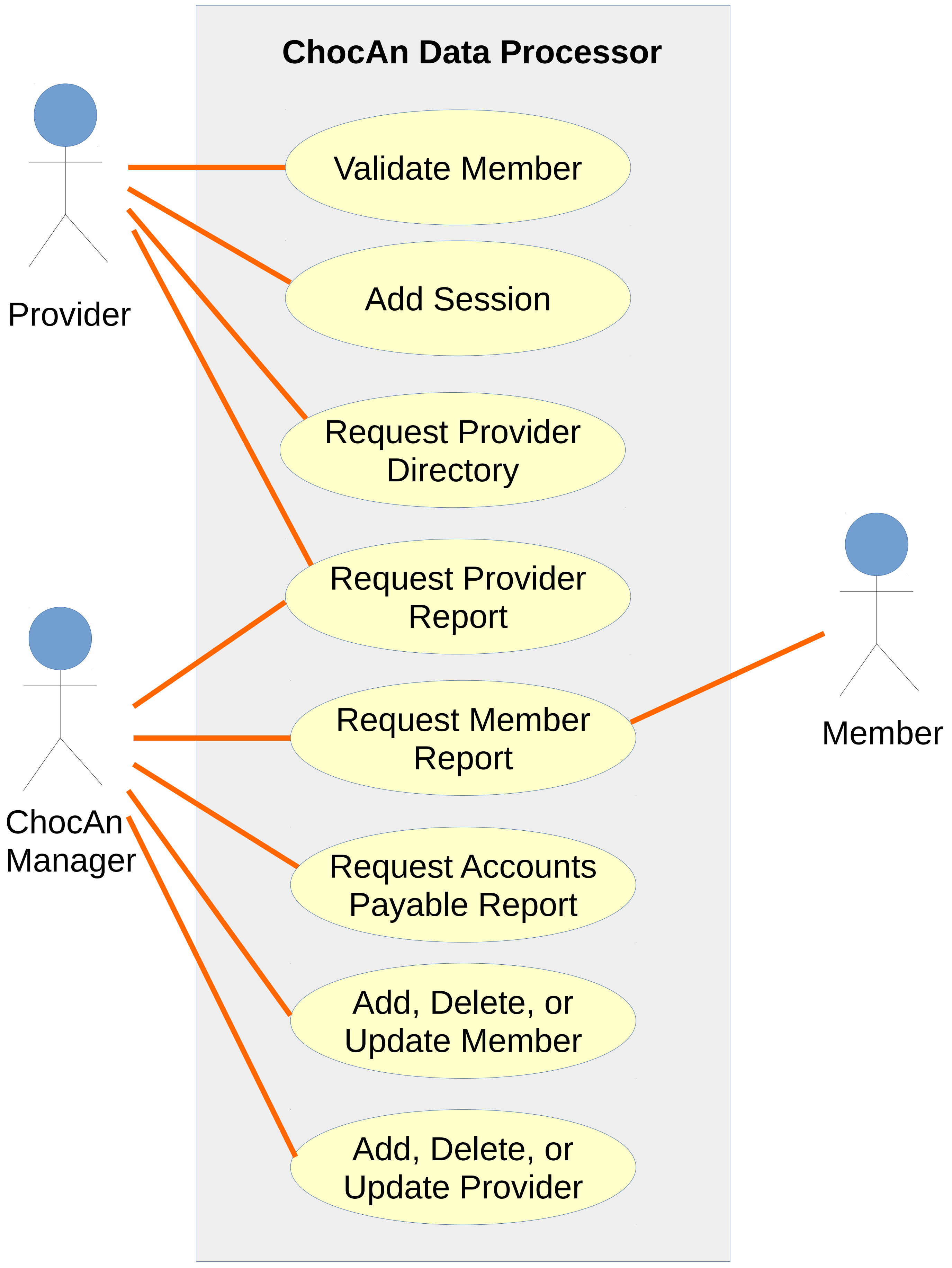
11.24 Perform the requirements workflow for the Chocoholics Anonymous project in Appendix A.

The first step: Glossary

This glossary is compiled from the appendix, and with terms of my own creation. Let's just pretend that they were obtained by interviewing the client and providers.

Accounts payable report: weekly report provided to ChocAn managers listing every provider to be paid that week, the number of consultations each had, and the provider's total fee.
Acme Accounting Services: third party organization responsible for financial services. Updates member records each day at 9 p.m.
Card: magnetic swipe care provided to members that is used for identification purposes. This card is embossed with the member number.
Card reader: device through which the card is swiped to validate the member.
Comments: optional comments in a Session record, with a maximum of 100 characters.
Date: a month, day, and year in the format MM-DD-YYYY.
Date service provided: the date on which a session occurred.
Date service recorded: the date on which a session was recorded by the product.
Interactive mode: mode in which the product operates during the day. Allows ChocAn employees to add, delete, and update member records.
Member: a ChocAn customer who purchases Provider services.
Member number: a unique 9 number code identifying a member.
Member report: a weekly report provided to members if they have used a service that week. Sessions are listed by order of date.
Provider: a health care professional who provides services to ChocAn members.
Provider directory: directory by which the provider looks up service names, codes, and fees.
Provider name: the name of a provider in 25 characters.
Provider number: a unique 9 number code identifying a provider.
Provider report: a report provided to providers on a weekly basis or upon request. The provider report is sent to the provider as an email attachment. The provider report lists the services provided by the provider for that week. The sessions are reported in the same order in which they were recorded to assist the provider with verification.
Service: the service provided by a provider (for example, dietician service).
Service code: a unique six number code identifying a service.
Service name: the name of a service in 20 characters.
Session: an instance of a service provided by a provider to a member.
Session record: record of a session containing the following information: date service recorded, date service provided, provider number, member number, service code, and comments.
Suspended: indicates that a member number applies to a ChocAn member who's account is suspended due to owed fees.
Terminal: computer terminal with which the provider interacts with the product.
The product: the system that ChocAn has contracted to be created.
Time: an hour, minute, and second in the format HH:MM::SS.
Validated: indicates that a member number applies to a ChocAn member who's account is not suspended.

The Second Step: Business Model



This step involves creating an understanding of the business processes through interviews and questionnaires. This is just an initial idea of the processes: for example, there may be situations where managers need to validate members, add sessions, and request provider directories. Based on the appendix, we have the following use cases:

Brief Description
The *Validate Member* use case enables a provider use a terminal and customer's card to ensure that the customer is a ChocAn member who's account has not been suspended. If validation fails, the terminal displays the reason: Invalid Number or Suspended.

Step-By-Step Description
1. The terminal dials the ChocAn data processor after the provider has swiped a member's card.
2. The terminal sends the 9 digit member code from the card to the ChocAn data processor.
3. The ChocAn data processor checks member records for a member with that member code.
3.1. If there is no member record with that member code, an INVALID NUMBER code is sent to the terminal.
3.2. If there is a member record with that member code, it's status is checked.
3.2.1. If the member account is currently suspended, a SUSPENDED code is sent to the terminal.
3.2.2. Otherwise, a VALIDATED code is sent to the terminal.

Brief Description
The *Add Session* use case enables a provider to add a session for a member and service.

Step-By-Step Description
1. The provider must first validate the member: this use case follows from the *Validate Member* use case returning the VALIDATED code to the terminal (3.2.2 of the *Validate Member* use case).
2. The provider uses the terminal to enter the date of the session in the format MM-DD-YYYY and the 9 digit provider code for the service provided, and sends this data to the ChocAn data processor.
3. The ChocAn data processor checks the provider records for a provider with that provider code.
3.1. If there is no provider record with that provider code, and INVALID PROVIDER code is is sent to the terminal.
3.2. If there is a provider record with that provider code, the ChocAn data processor returns the name of that provider to the terminal for confirmation.
3.2.1. The terminal prompts the provider for confirmation that the name matching the provider code matches that of the provider they specified.
3.2.2. If the provider does not confirm the provider name, the ChocAn data processor cancels the transaction.
3.2.3. If the provider confirms the provider name, the terminal sends additional comments to the ChocAn data processor that were entered by the provider.
3.2.3.1. The ChocAn creates a session record with the following data:
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 (100 characters) (optional).
3.2.3.2. The terminal displays the cost of the service for confirmation.

Brief Description
The *Request Provider Directory* use case enables a provider to request a copy of an up to date provider directory.

Step-By-Step Description
1. The provider requests an up to date provider directory through the terminal.
2. The terminal connects to the ChocAn data processor and sends the provider code and the request.
3. The ChocAn data processor validates that the provider code matches one found in a provider
4. If a provider record is not found with a provider code matching the one given, return an error.
5. If a provider record is found with a matching provider code, compile a report of all services, alphabetically, as name, code, and fee.
6. The compiled provider directory is emailed as an attachment to the provider's e-mail address.

Brief Description
The *Request Provider Report* use case enables a provider to request an up to date provider report.

Step-By-Step Description
1. The provider requests a provider report through the terminal.
2. The ChocAn data processor finds the provider record for the requesting provider, and adds the following information to the start of the provider report:
    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).
3. The ChocAn data processor searches all session records for the current week with a provider code matching that of the requesting provider.
4. The list of session records are sorted by the date of the service provided.
5. The session records are appended to the report in the format:
    Date of service (MM–DD–YYYY).
    Date and time data were received by the computer (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).
6. The number of sessions found is appended to the report.
7. The total of the fees of the found sessions is appended to the report.
8. An EFT record is written to disk with the information required to pay the provider the appropriate amount (further details require consultation with the accounting company).

Brief Description
The *Request Member Report* use case enables a member to request an up to date member report.

Step-By-Step Description
1. The member requests a member report by e-mail or through an online service.
2. The ChocAn data processor searches all member records for a record with a member code matching that of the requesting member.
3. The member report is started with the following data from the member record:
    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).
3. The ChocAn data processor searches all session records for sessions for the current week with a member code matching that of the requesting member.
4. The session records are appended to the member report as:
    Date of service (MM–DD–YYYY).
    Provider name (25 characters).
    Service name (20 characters).

Brief Description
The *Request Accounts Payable Report* use case enables a ChocAn manager to request an up to date accounts payable report.

Step-By-Step Description
1. The ChocAn manager requests an accounts payable report through the manager interface of the ChocAn data processor.
2. All EFT records are searched, and sorted into separate lists by provider.
3. For each provider, append to the report the name and provider code of the provider, the number of sessions (the number of EFT records for that provider) and the total of the fees of all EFT records for that provider.
4. The sum of all unique providers found in all EFT records is appended to the report.
5. The sum of all sessions found in all EFT records is appended to the report.
6. The sum of all fees found in all EFT records is appended to the report.

Brief Description
The *Add, Delete, or Update Member* use case enables a ChocAn manager to add a new member record, delete a member record, or update a member record. (Perfect opportunity for informational coherency).

Step-By-Step Description
1. The ChocAn manager accesses member services through the ChocAn data processor.
2. The ChocAn data processor prompts the manager with options to add, delete, or update a member record, or go back.
3. If the manager chooses to go back, the ChocAn data processor returns to the previous screen.
4. If the manager chooses to add a new member, the manager is prompted to provide the information required for a member record:
    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).
    Member e-mail address (25 characters).
    4.1. The member record is written to disk with a generated member code (9 digits). Do not check for accounts that are identical in all ways except for member code: it's not completely impossible for two people with the same name to reside at the same address.
5. If the manager chooses to modify a member, the manager is prompted to enter the member code.
    5.1. The member records are searched for a record with a member code matching the one provided.
    5.2. If a record with a matching member code is not found, the manager is prompted with an error message.
    5.3. If a record with a matching member code is found, the record data is displayed on the screen for modification.
        5.3.1. When finished modifying the member data, the manager submits the data, and the member record is rewritten with the new data.
6. If the manager chooses to delete a member, the manager is prompted to enter the member code.
    6.1. The member records are searched for a record with a member code matching the one provided.
    6.2. If a record with a matching member code is not found, the manager is prompted with an error message.
    6.3. If a record with a matching member code is found, the record is NOT deleted, but is moved to a repository for long term storage (It may be necessary to keep these records for later reactivation or for legal or accounting purposes).

Brief Description

The *Add, Delete, or Update Provider* use case enables a ChocAn manager to add a new provider record, delete a provider record, or update a provider record. (Perfect opportunity for informational coherency).

Step-By-Step Description

1. The ChocAn manager accesses provider services through the ChocAn data processor.
2. The ChocAn data processor prompts the manager with options to add, delete, or update a provider record, or go back.
3. If the manager chooses to go back, the ChocAn data processor returns to the previous screen.
4. If the manager chooses to add a new provider, the manager is prompted to provide the information required for a provider record:
   - 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).
   - Provider e-mail address (25 characters).
   - 4.1. The provider record is written to disk with a generated provider code (9 digits).
5. If the manager chooses to modify a provider, the manager is prompted to enter the provider code.
   - 5.1. The provider records are searched for a record with a provider code matching the one provided.
   - 5.2. If a record with a matching provider code is not found, the manager is prompted with an error message.
   - 5.3. If a record with a matching provider code is found, the record data is displayed on the screen for modification.
     - 5.3.1. When finished modifying the provider data, the manager submits the data, and the provider record is rewritten with the new data.
6. If the manager chooses to delete a provider, the manager is prompted to enter the provider code.
   - 6.1. The provider records are searched for a record with a provider code matching the one provided.
   - 6.2. If a record with a matching provider code is not found, the manager is prompted with an error message.
   - 6.3. If a record with a matching provider code is found, the record is NOT deleted, but is moved to a repository for long term storage (It may be necessary to keep these records for later reactivation or for legal or accounting purposes).