# HEMP *light*

## Part of the HEMP Enterprise Software Analysis and Design Series

(Holistic Enterprise Mechanization Process)

by David E. Jones

Last Revised 20 Jun 2009

 

# Table of Contents



 

# Introduction to HEMP *light*

HEMP is a holistic process and series of artifacts used to drive requirements analysis and solution design in preparation for development and customization of enterprise systems. It is holistic because the point is to gather, organize, and consider all relevant requirements (and sometimes less relevant ones) and use them as a basis for the designs that the system will be based on. HEMP *light* is a simplified version of HEMP that is meant for smaller or less formal projects with fewer people involved.

If there are more than a few people involved with specifying requirements or collaborating on designs then this simplified version of HEMP may leave too many details undocumented and that will cause problems, especially in the design and implementation efforts. Those are cases where the more formal and complete documentation in HEMP Complete, such as use cases and data statements and test scenarios, are well worth the extra effort required to create them.

One scenario it is good for is a customization effort where a system will be used mostly as-is with things added and changed as needed to fit the requirements of the specific end-user organization. This often involves writing a story (or editing an existing generic story such as those from the Apache OFBiz Universal Business Process Library), doing an overlap and gap analysis to see what the system covers, and then creating designs to fill in the gaps found. When this changes from customizing an existing system to designing and building a new system (even if based on or reusing existing pieces) then HEMP light may not be adequate and HEMP Complete is recommended.

The idea of the general process is to start with requirements and move toward designs on which the implementation can be based. Each set of artifacts produced along the way is meant to lead into the next with an incremental effort that is "easy" when done based on the previous artifacts.

This takes into account making it easier to change things early on where that change is cheaper and easier. As these are established and discussed at each stage it should help those involved to focus on one aspect of the requirements at a time and move from more general to more specific as incremental decisions are made.

It also takes into account rework as needed based on prior artifacts. For example once wireframes are created it is possible to do some role-play style testing with end-users. Based on the feedback from this testing it may be possible to improve the system by changing the user interface. This is much easier to do when you can refer back to the requirements that preceded the design so that new designs can still be based on and reviewed against those requirements. This opens up a lot of flexibility during the design process to change things and still satisfy the goals and needs that prompted the automation effort in the first place.

## The Story of HEMP *light*

The story below is an example of the type of story that will be written to gather and organize requirements. This story corresponds to the artifact flow diagram that is on the last page of this document.

This short story combines high level process steps with the details for those steps. For larger stories the more detailed steps are often best kept in separate documents and then the high level story can tie it all together with sentences that link to documents

 

containing the lower level stories. The Apache OFBiz UBPL is a good example of how this can be done.

One thing to note is that while it is not recommended to mention system interactions in a story (because that is a design and not a documentation of a requirement) the word “system” is used frequently in this particular story. The difference is that the system referred to here is the system that will be built by following the story, and not a system that would facilitate in performing this story. The HEMP process could be followed to take this story and then design and implement a system that does help automate this process.

And the story begins...

**Expert User gathers Requirements**: Expert User writes Process Story. While writing Process Story, Expert User finds requirements that do not fit as part of a story (ie are not part of any process). For each requirement not part of any process Expert User writes a Requirement Statement.

**Analyst prepares Initial Design**: Analyst reviews each step (business activity) of Process Story. If the process step is already handled by the existing system Analyst documents how that activity is preformed in the system. If the process step is not handled by the existing system Analyst documents the gap and then describes how the user should be able to interact with the system in order to handle the process step.

**UI and System Designers formalize and detail system Design**: UI Designer reviews all gaps in the Overlap and Gap Analysis and based on them writes a Screen Outline for each new screen or screen change described in the user interactions to fill the gap. For complicated screens where the layout is unclear from the outline, UI Designer creates a Functional Wireframe to make the layout clear. System Designer reviews all gaps in the Overlap and Gap Analysis and creates or extends a Data Model that has a place for all data mentioned in the gaps. System Designer prepares Initial Data based on gaps and on the Data Model to demonstrate usage of the model, configure settings, and so on.

**Developer creates Implementation based on Design**: For development in Apache OFBiz Developer creates Controller Request and View entries, Menu definitions, Screen definitions, Form definitions and Template implementations, and Service definitions and implementations, all according to details in the Design. For developments on other frameworks Developer creates whatever artifacts are necessary for screens, background processes, and so on to implement according to the Design.

 

# Gather Requirements

## Process Story and Experience Story

Process Stories are a simple way to document a series of business activities. They should be written as active verb phrases with each one stating who (actor) does what (action). The story is written with one activity following another in order of dependency where relevant, and in logical ordering or grouping otherwise.

The Stories (aka Narratives) should be process oriented and very informal. They should change frequently as discussions progress and capture everything related to business processes and business strategies. To get started a high level list of business strategies and goals may help to organize things and lead to documentation of what needs to be done in order to reach those goals and pursue the strategies.

In order to focus on requirements and avoid premature design the Stories should not mention anything to do with “the system” or with system interactions. Consider that systems are just tools for communication and coordination and all processes can be expressed in terms of actors in different roles working with other actors. It is very natural for ideas about system design and other things not relevant to the Stories to come up while writing them. This is a good thing and these are very valuable for the eventual system design and implementation. The best thing to do with these ideas is to docment them as Requirement Statements in order to remember them and make sure they are represented in later artifacts.

On a side note, while you should avoid talking about the system because you may cross the line into starting design on the system, you should talk about other systems that exist and will continue to be used. Treat existing and external systems as Actors like any other and describe the actions they perform and how other actors will interact with them. When describing how other actors will interact with system actors try to do so in a technology neutral way with details about what they do and what information is moved around, and not about how it is done or how the information moves. Those “hows” are the design that will be based on these requirements of “whats”.

It is easy to see these as of little value because everyone in the organization already knows what the high level processes look like. As a group goes through the process of writing these stories one of the most valuable outcomes is to realize that different people understand the processes and priorities very differently and this effort gives everyone a chance to get on the same page, while at the same time establishing an overall structure and set of goals for the eventual system to meet. These Stories represent a certain form for requirements that translate well into use cases and system designs, and are often more difficult to write and make complete than one would think getting started with them. They really are the opposite of simple and useless being a rich and well-organized way to document nearly all important requirements.

Initial stories should be from the perspective of the organization, and again, with the focus on process. In general each story should be no more than a few pages of text with detail broken down into sub-stories as needed. The top-level story describes everything the business does. Individual sentences in the top-level story can be expanded in more detailed “sub-stories”, and going down multiple levels as needed to provide full detail.

These Process Stories are basically a series of sentences describing each process step in terms of who does what. However, they are meant to be general information and

 

comments about an actor's thoughts and goals can be valuable. The lowest level stories (the "leaf" stories in the hierarchy) should include around 5-15 steps (sentences), which is usually a good number of business process steps to include in a use case (which may eventually be based on these Stories). If you go too much over that look for easy hand-off points where the story can be broken up into multiple stories, and remember to change the level above to represent the divided story.

The Process Stories described above are organized top-down, and that is a good way to write them too. Start with a top-level story that describes general processes that the business goes through in order to meet its strategic objectives. If the strategy for the company is not well defined you may need to do that first in order to have something to refer back to while making decisions about this or that way of doing things. In other words when you have a good set of strategic objectives defined you can ask which option best meets those objectives, and if none do keep working until one is defined that does a good job.

These Process Stories should focus on the "success" or "happy" scenarios, but especially as lower level (more detailed) Stories are written it is good to consider important exceptions and alternate flows. Still, to remain focused when writing Stories in groups it is helpful to focus on the main success path in order to make progress and not miss important parts of the processes, especially in higher level (less detailed) Stories. As Process Stories get more detailed alternate scenarios should be included in the flow (if needed they can be separated or organized later while preparing for transition to the more formal Use Case documents).

The first pass on Process Stories should try to document what is happening in the company as it is now. If there is no desire to change how the company is operating then things can stay in this form. It is more common that changes in the company's processes will be desired. Once the current processes are documented in story form it is an excellent starting place and a tool to facilitate discussion about changes in the process and document them as discussions progress. Some process improvement efforts may be difficult when working only from these Stories and things will likely come up when working on the Business Use Cases related to the Stories that will feed back into more changes to the Stories.

Because Process Stories are from the perspective of the organization (as opposed to Experience Stories which are from the perspective of a single Actor), questions relevant for them include: What happens in the organization? Who does what? When something happens what other things does it make possible? In order for something to happen is there anything that must happen before? How does each Actor know when they should do something? When and how does an Actor pass the baton to another Actor? How does an Actor decide which other Actors to involve or which other processes need to happen?

One common point of confusion is how to write about existing systems that will stay in place. One of the “rules” about writing Stories is to not mention “the system”, but that just means the system to be built and not any existing systems. Existing systems should simply be treated as Actors that perform actions and that other actors interact with. Anytime an Actor interacts with another Actor that represents an existing system you know that an integration between the systems is needed.

Once the Process Stories from an organization perspective are in place it is easy to identify a list of Actors and define each of them. See the Actor Definitions section for

 

more details. It is also easy to identify the functionality needed based on the process perspective stories.

Another form of story that can be valuable is a story from the perspective of a specific Actor. This facilitates gathering details about the concerns of each Actor through a "day in the life of..." type of format. Questions relevant for Actor Perspective Stories include: What are the goals of the Actor? What is the Actor trying to do? How will the Actor go about getting those things done and making sure they are done correctly? What resources and constraints affect the Actor? How does that Actor interact with other Actors and with the Organization?

**Pitfall: Actor/Role-based Organization**

Starting with stories from an Actor perspective and organizing the stories (or other requirements) by Actor typically leads to redundant and difficult to organize stories because the same processes will be described multiple times from different perspectives. It can also increase the chances that important processes or Actors are not discovered and documented until much later because the stories are derived from the actors, instead of the actors from the stories. **Just remember** that when you write in terms of process it is easy to figure out which Actors there are and what they do, but if you write in terms of Actors it is hard to figure out what the overall process is. In other words, you can derive actors from a well written Process Story, but you can't reliably derive the overall process from even the best written Actor stories.

**Pitfall: Feature-based Organization**

Even more problematic than starting with actor perspective stories is starting with functionality or feature perspective stories and organizing stories and other requirements by features. The biggest problem with this is the tendency to skip the requirements gathering effort and jump right to designing solutions. When choosing how to organize of documents remember that features generally have many touch points in different business processes so starting by writing stories from a feature perspective causes similar problems to starting from an actor perspective, but typically on a larger scale with more disconnects causing redundant processes and disorganized requirements. **Just remember** that as with Actors when you organize by functionality it is hard to figure out the processes and the Actors, but when you organize by process and you know what needs to happen you can more easily design the functionality to make those things happen.

**Pitfall: Headings/Topics First, Detail Later**

It may be tempting to try to think in advance about what the process your are writing a story for will consist of and create an outline for it, and then fill in the detail of each section. That may be a fine way to write a book or organize things already well established, but when writing stories it is best to leave things open and let them shake out as they will, and don't assume to much in the beginning. Take things step my step and write each step in terms of actor and action (who does what). The biggest problem with structuring the document first is that by defining a structure you create a box and then you tend to get stuck in that as you think about the process. Anything outside of the boxes you've defined, or that crosses between two boxes, may never occur to you or might be described poorly as you try to work around the boxes you have created. **Just remember** it is easier to figure out the structure of a process once you have the process documented and can review it an find clean hand-off, isolation, and frequency mismatch points in the process.

 

**Pitfall: Documenting the “Why” instead of the “What”**

Don't document the “why” as it is too subjective and shifty. Don't try to document rationale, or the why of doing these things. It is a trap where you spend a lot of time and get nothing out of it.

It's not that you shouldn't discuss why to do things, just don't try to document it. Basically documenting the activities gives a concrete framework for discussion and understanding around why things are done, and discussion around why things are done leads to decisions about what to do and the "what" is much easier to document.

The goal is to gather relevant information for implementing a business automation system as quickly and effectively as possible. You could spend all day talking about why, and if you focus on that instead of what then chances are you WILL spend the whole day talking about why and it will waste time and make your client and others feel very frustrated. This frustration part is a big deal. Often people get rather emotional about how they run their business and if you talk about it in terms of why it is too easy to get personal (as it is VERY subjective), and too hard to see the real point. If you talk about it in terms of what (the specific actions) then it is more objective and easier for everyone to see the impact of doing things a certain way, and how the decision to do something at one point in the process affects other things elsewhere in the process.

In some situations you won't run into discussing the “why” very much because good planning and preparation has been done around the processes, and there are only a few places you'll have to dig into and refine processes to make them concrete enough to automate. In other situations it is a very different story. Most of the time may be spent discussing just a few topics because there are issues in them that people are very frustrated and emotional about. You can talk a little bit about why things were done, but it often turns out that the reasons for doing certain things in a certain order had nothing to do with the business goal, and people won't realize it until you look into detail at what was being done and forget about why. Just stick to talking about the activities and how they are ordered and what leads to them being ordered that way. In the end you'll hopefully find a much more effective way to handle what is needed, and result in a process that should avoid the "majority of the problems that were taking up the majority of the time."

For example one company had shipping estimates that were too low and in their process they didn't find out that was the case until **after** the credit card was charged. They were able to identify that problem and the pain it caused, and when asked why they were doing things in that order (that was causing the problem) they were stuck on the process they had. This included verifying the pick before doing packing which was such an important aspect of what they do that they couldn't even discuss doing anything in a different order because that "why" was so important to them. Once the conversation focused on what was being done and when, they were able to clearly see the cause of the problem, and see that we weren't talking about having to eliminate the verification station, it was just necessary to move the changing of the Shipment status to "Packed" to the packing station when they had weighed the package(s) and had a shipping quote so they could handle low shipping estimates before the credit card was charged.

## Actor Definition

Actor definitions should include a description of each Actor mentioned in the stories. Actors may be users, groups of users, or even systems. Understanding the actors helps

 

to organize the processes and provides important information about the business activities each is related to.

Once defined, stories from each actor's perspective can be written. As described in the stories section these can give different perspectives and help define motivations of different actors related to the overall effort.

### Requirement Statement

While Stories are valuable for organizing a business and the systems that help automate it, they are not good for capturing everything that is important to consider when designing and building the systems. There are many requirements that do not fit well into Stories, or that apply to all Stories.

The most common form of Requirement Statement is to capture business requirements that apply to all processes and so don't belong in any particular one. When working with this sort of Requirement Statement make sure it really can't be incorporated into stories since that is a much easier and more naturally organized way of managing requirements.

Another form of Requirement Statement is to capture thoughts about tools and system features and how users might interact with the system, as opposed to Stories which should avoid these things.

An alternate use of a Requirement Statement is to keep track of ideas that come up in discussions but that you are not ready to incorporate into stories yet. These “Ideas to Incorporate” become a simple to-do list and help you make sure things are included without having to pause to actually include them. This is especially helpful when an idea comes up that may have many different touch points in different stories, so some real planning and discussion will be needed before the idea will be fully incorporated.

Requirement Statements are intentionally unstructured and are meant to be an easy way to capture thoughts while brainstorming or while writing Process Stories so that you don't have to interrupt the effort of writing the stories. Again these are very informal ways of documenting requirements that will be incorporated into more formal artifacts later on, especially when doing System Interaction Use Cases, Screen/Report Outlines, Data Statements, and other design artifacts.

Like Stories, Statements can be written from different perspectives and with aspects of the eventual system in mind. One common type of Statement is the Ability Statement, usually used to describe desired abilities of the system itself. They can also be used to define abilities of Actors, including users and other systems.

Because of the nature of requirement statements you usually won't have too many of them, but you may end up with a large number. It is important to keep them organized as you go, keeping similar and related statements close together, and even creating categories of statements to help organize them better.

While reviewing and organizing Requirement Statements check each one to see if it really can't be incorporated into the Process Stories. It is common that a requirement is thought of as not being part of any process because it touches multiple processes. In that case, it is far better to adjust each relevant process so that the requirement is not forgotten when designs are created based on that process.

When working on statements don’t try to make them “complete.” The purpose of the Process Stories is to help insure that everything is considered and documented

 

somewhere. The purpose of Requirement Statements is to make sure that ideas which are not appropriate in Process Stories have a place to live until they are incorporated in the System Interaction Use Cases and later design artifacts.

**Pitfall: Redundancy and Inconsistency in Statements**

Because statements are usually a result of brainstorming and general ideas that come up while writing stories, the same ideas will probably come up over and over. It is easy to create multiple statements that mean the same thing, or even worse something similar but that contradicts the other statement(s). In fact, by nature redundancy tends to lead to inconsistency and that inconsistency is something that must be clarified or resolved before the requirements can be considered ready for the next step. While redundancy alone leads to inefficiency, inconsistency leads to confusion and can result in more inconsistency later on that cause problems with design and possibly the usefulness of the system too. Keeping statements organized and regularly reviewing and refining the statements as you go, and moving them to Process Stories when possible, can help reduce or eliminate these issues.

 

# Initial Design

## Overlap and Gap Analysis

The Overlap and Gap Analysis artifact is based on the Process Story and the Requirement Statement artifacts. It is structured the same way as the documents, in a very literal way, since each Overlap and Gap Analysis is simply a response to a particular Process Story sentence (process step) or Requirement Statement.

In this artifact where there is an overlap the existing system elements that cover the requirement should be documented in detail, even in the form of step-by-step instructions for how a user or external system might interact with the existing system to perform the business activity described in the process step.

When there is a gap this artifact should document any partial overlap, and then go on to describe general system interactions that the system would need to support in order to fill the gap and satisfy the requirement. This is the most important part of the Overlap and Gap Analysis artifact because it is a sort of Initial Design that leads to the more detailed UI and system designs that lead to the system implementation. Once the implementation is done (or at least the detailed design is done) you can go back to this Overlap and Gap Analysis artifact and change the corresponding gap documentation to represent an overlap instead.

The goal of this artifact is to eventually document how the system can be used to meet each requirement represented in the Process Story and Requirement Statement artifacts.

This artifact can be a copy of a Process Story or Requirement Statements document with the overlaps and gaps documented inline, or it can be a number of small documents (one for each Process Story sentence or Requirement Statement) that are linked to from these other artifacts. It is best to keep it separate as you may have different Overlap and Gap Analysis artifacts done to compare the requirements to different existing systems, or to explore different designs, and getting back to the goal of this artifact you would also want to have the 100% overlap document maintained separately from the original Overlap and Gap Analysis.

 

# Design

## Screen and Report Outline

This outline is created by reviewing all system interactions described in the Overlap and Gap Analysis and making sure there is a screen with adequate elements for each interaction. The outline node for each screen should include information about what is presented to the user, what is solicited from the user, and all of the exit points from the screen. These should include links and form submissions, and to what other screens each exit point can lead to. If there are multiple screens that a single exit point can lead to it should describe how the target screen is decided. This introduces some notion of flow between the screens, and that will feed into the screen flow diagram.

For reports in the outline the structure is very similar. Some reports are meant to be part of a process and have more tactical information. Reports with more strategic information may exist outside of the normal business process flow and can be listed in a separate part of the outline.

The Screen/Report Outline is a separate document from the Overlap and Gap Analysis because it is structured completely differently. A Overlap and Gap Analysis is organized by business process but the Screen Outline will be organized by screens and what is on them. Each screen may be involved in various processes or parts of a process, and conversely each process may be automated with multiple screens.

Each item in the outline should refer back to the gap description (in the Overlap and Gap Analysis) and business process steps or system interactions that it came from. While these references to source information are optional, if you run into problems with consistency between Gap Analysis and Screen Outlines, or with designs in the Screen Outline that don't seem to be based on any requirements or preliminary designs, you might want to consider using this technique or requiring this from your team.

A rough structure for a Screen Outline might look something like:

1. Screen Foo Baz
    - 1.1. Area 1 (top of page)
        - 1.1.1. Static Text XYZ
        - 1.1.2. Dynamic info foo
    - 1.2. Area 2 (bottom of page)
        - 1.2.1. Dynamic title text baz
        - 1.2.2. Form for baz
            - 1.2.2.1. Field 1 - text box
            - 1.2.2.2. Field 2 - drop-down
            - 1.2.2.3. Submit Button (goes to Screen Bar Foo)
2. Screen Bar Foo
    - 2.1. et cetera...

The outline should be created by going through each gap description's system interaction and creating a place for it on a screen. There will likely be a refinement process that will require you to move things to different screens, combine or split screens, and so on, as further system interactions are reviewed. Just like with most of

 

these intermediate artifacts the point is to have something that is easy to create and change as discussions continue and as decisions are made.

Once the outline is initially complete and all system interactions have been incorporated it is a good idea to check the design by doing some "role playing". This can be a quick sanity check to just walk through the processes and make sure each thing described has a place and is easy to find and use, or more to the point makes it easy to accomplish the goal defined in the process. Once wireframes are created from the content of the outline more comprehensive role playing can be done with prospective end-users, but for now some informal stuff among the designers is enough to help make sure that nothing was missed and that everything makes sense, as well as create further opportunities for design ideas and better ways of organizing the screens and what is on each.

### Functional Wireframe

Functional Wireframes should be created for anything that a user will see, including screens (web pages and desktop application screens), reports, emails, and so on. While the Screen/Report Outline contains much more detail, and should be used as a supporting document for the Functional Wireframes, it is difficult to represent spatial relationships and layout in a text document and a wireframe is great for expressing that literally. The Functional Wireframe is created using the information in the Screen/Report Outline.

Functional Wireframes are also useful for testing with prospective end-users. A simple way to do this is by role playing. This is kind of like you did in high school (or if you were cool then starting in elementary school and going through and past college...), but way more boring since you'll pretend to be either a part of an ERP system or a user of the system instead of a dwarf, elf, or eog armored super-hobbit. To do this have one person (typically an analyst or designer), or even a few people, pretend to be the system and hold the functional wireframes. The prospective end-users will pick an Actor and play that character. While doing this the person playing the new system should only show the wireframe and not try to describe what the user should do or how they should do it, but let the person playing the user figure things out. The user person describes what they do, and the system person describes how the system responds. Both have the benefit of having read the documents related to what they are doing, and the system person should follow this and the design documents carefully, but the user person should not refer to them or try to follow them and instead try to go about their daily activities or work toward specific objectives.

To create a Functional Wireframe you can use a more formal approach like a diagramming tool, or even just sketch it on paper or whiteboard (and scan it or take a picture of it to digitize it if you need that). A paper sketch is a nice way to do a rough pass, but they are difficult to change. While a diagramming tool may seem more cumbersome they are certainly easier to change. There is also the benefit that the results are nicer looking and often more clear with a diagramming tool, but except for in the largest of groups that is secondary to the benefit of being easy to change. My personal preference is to rough out an initial layout on a whiteboard and once things seem well enough organized create a cleaner wireframe using a diagramming tool.

The process of creating a Functional Wireframe from a Screen Outline is pretty simple. Just go through each of the visual elements in the outline and draw them in the wireframe. Remember to include references back to the Screen Outline item that the screen element came from. When using a diagramming tool that supports layers it is

 

nice to put this information in a separate layer that can be overlaid on top of the main wireframe, usually with transparent text and shading so that the main wireframe is still easy to see. For some screens simple is definitely the right word for what needs to be done! Creating a layout that works and makes sense for screens with lots of elements or complex interactions can be a lot of work. Some wireframes may require only a few minutes of work, and others may take days of effort and lots of review with different people.

In some cases a number of wireframes will need to be created because different parts of the screen may expand and collapse or popups may appear, or the options in an area may change as other things are selected on the screen. It is generally easier to create multiple wireframes than to try to express all of this in a single one. Another thing to consider is a sort of “flannel board” approach where parts of the screen are blank in the wireframe and there are various versions of that part of the screen that can be dropped in or attached, like on a flannel board, to that part of the main screen.

While trying to lay things out visually you may find that things described in the Screen Outline won’t actually work on a two-dimensional screen of finite size. Feel free to change the Screen Outline as needed while going through this. In fact, make sure to because you don’t want these to become inconsistent. In some cases you may even end up reworking the screens enough to require changes to the system interaction descriptions in the Use Cases, and then corresponding changes in the Screen Outline. Just remember that this is a process of refinement and you are looking at the system from different perspectives as you create different artifacts. Also remember that it is still pretty easy to change things now and that as you get into more formal artifacts it will become more difficult and expensive to change things.

### Data Model

The Data Model is what will eventually be used to physically persist information that the applications need in order to function. The model document described here is a relational database model. While other means of persisting data exist, relational databases still drive the bulk of the business world and they are likely to remain so because no other persistence technology can match it in terms of flexible storage and retrieval and flexible searching and querying. If a different means of organizing and persisting data is desired (object database, directory/hierarchy datastore, ontology server, XML document store) then an artifact that better suits it can be substituted here.

The data is vitally important because no matter what the system does as it operates eventually every system operation will finish and all that will be left to feed into other operations is the persisted data, either in the main system or in other systems it integrates with.

While creating a Data Model is a non-trivial task, representing one is pretty simple. The most important elements are entities (tables) and fields (columns). In a first pass organizing all of the information into entities and fields is plenty to do. Another important part is specifying unique identifiers (primary keys) and relationships to other entities (foreign keys), and you may want to do that in the initial pass or at least just after it. After getting those in place adding more detail like data types and sizes for each field can be done.

The Data Model is most easily and effectively created based on the Data Statements which describe the nature of, and relationships between, the data that the system needs to keep track of. As with other artifacts it is important to refer back to the artifacts it

 

came from. In HEMP light where Data Statements are not used simply base the Data Model on the mention of information to track in the Overlap and Gap Analysis.

Experience with data modeling patterns and techniques is vital to getting a good result, and there are lots of good patterns, and lots of debate about what makes a good pattern. With modern systems in order to keep things flexible and to reduce the need for changes in the future it is usually best to err on the side of normalization and split fields into different tables whenever there is not a really clean one-to-one relationship between the fields.

When working on a Data Model that will be incorporated into an existing system it is important to make sure the data model incorporates elements of the existing system and reuses parts of it as much as possible, and that it extends the model where needed while maintaining connections to it in order to avoid ending up with an independent and disconnected data model that limits flexibility and system capabilities. If you know that a certain set of Data Statements (or mention of information to track in Overlap and Gap Analysis) maps clearly to parts of the existing data model then you can just refer to those in this Data Model document.

If you think that a set of information to track might map to one or more parts of the existing data model but you aren't sure, then write it out as a set of new entities in the Data Model document and after it is complete compare it to existing structures to see if there is a match now that they are in a more similar form. In other words, based on Data Statements or mention of information to track alone it can be difficult to see if things match well or not, but after looking at a "clean room" layout of the new data model it will be easier to compare it to an existing model.

### Initial Data

Other terms often used for Initial Data, and different types of Initial Data, include seed data, demo data, and test data. For any of these the purpose of the data is to supplement the Data Model in order to help describe different meanings for data and different options available, or to communicate better how the Data Model is meant to be used by giving real world examples of what might go in each field.

Seed data is data that it maintained with the code and that code may rely on (including unique identifier direct references) in order to operate. There are many varieties of seed data including: statuses, enumerated options, and type options to allow common data structures to be reused (such as Sales and Purchase Orders which share many but not all of the same fields and related entities, and which are generally treated very differently by code). In general seed data should be updated when the code of a system is updated to make sure the latest code has the options in place that it expects.

There is one exception with seed data that is required for operation but that should only be loaded once because either the system or a user is expected to change it. One example of this is a background process definition that goes into the database that the system changes as it runs that process over time. You don't want another process introduced or that process reverted to its original state when the seed data is reloaded after a code update.

Demo and test data have different purposes, but the same data can generally be used for both. Demo data is meant to get things setup to make is easy to see how different parts and features of the system function. This is most valuable when different combinations of data produce certain results. The Demo data becomes something that can be used through the user interface of an application to see what it does in different

 

circumstances, and also as a very concise and direct form of self-documentation that analysts, developers, testers, and others can use to better understand how things are structured and what sorts of options are available.

Test data is meant more for testing purposes and expands on the idea of Demo data because it may be significantly more redundant and voluminous in order to support a wide variety of usually automated test processes. Test data may certainly be used for manual test processes, and in that case it ends up looking a lot like demo data. So again these are two purposes for data but the same data can often be used in both ways with the special variation in form for test data used by automated tests. Both are useful for better understanding the system and documenting the intended state of data in different circumstances, or at different steps in the business processes (defined in the Process Stories) that the system helps automate.

The structure and form of these files varies according to what the system will support. It is possible to initially create the data in spreadsheet or very generic XML files if no target system or architecture has been defined yet, and then translate the data as needed when the system is eventually selected (never underestimate the power of a “mail-merge” from a spreadsheet to produce text in the form you want it!). Still, knowing how to format the data, and even better having a tool that can import it and check it against the Data Model definitions (preferably in a real database) is a great way to avoid simple errors that must be fixed before the Initial Data is really useable for any purpose.

 

# Implementation

Regardless of the tools and reusable implementation artifacts used, there are many aspects of implementation that apply.

Some tools are lower-level, requiring more development effort, and along with that often more flexible. For those tools the transition between detailed Design artifacts and the Implementation artifacts will not be as literal, or in other words there may be many options for the tool to use to implement what is documented in a particular detailed Design artifact. When this is the case, especially for larger projects, it is a good idea to plan in advance on which tools to use and how they will be used relative to the Design artifacts that need to be implemented.

With the Apache OFBiz Framework, and various frameworks like it, there are tools that correspond pretty well to typical design artifacts. In the HEMP Complete set of artifacts there are Design artifacts that correspond fairly literally to the Implementation artifacts. For HEMP light certain design artifacts are left out and so the implementation effort involves drawing from less literal sources and doing so on the fly instead of documenting it first, but the information in the end is the same.

Because Implementation involves the final check of detailed Design artifacts it is common for designs to be adjusted based on feasibility and level of effort estimates.

The basic responsibility of the Developer is to create Implementation artifacts based on the detailed Design artifacts, and then to make sure that the resulting system functions as described in the designs. Before the Post-Implementation steps can happen the implementation needs to be fairly bug-free, as the point of the Post-Implementation is to review for closeness to designs, satisfying requirements, and then applicability to the organization and the various Actors using it.

 

# Post-Implementation Review and Sign-Off

Just as the Design artifacts can be reviewed using the Requirement artifacts, the Implementation artifacts (the system itself) should be reviewed by Developers using the Design artifacts. In addition to this effort the UI and System Designers should also review what is implemented according to the detailed designs to make sure they were interpreted correctly.

To take it one step further and do a final check of meeting requirements and business applicability the Analysts should review the implementation against the Overlap and Gap Analysis they created, and also create the variation of the document mentioned above that has 100% overlaps documented. Once that is done the Expert User and other end-users can review and test the system based on the Requirements that they helped gather and document.

The general idea is that the same people involved with creating an artifact should review the final result against the artifact they created. In a way this means to just go backwards through the development process to review and test what was implemented. Each Actor involved represents a stakeholder in the effort and this will assure that they get a chance to sign-off on the results in a natural way.

When issues are found by an actor reviewing against a specific artifact they should first determine if the artifact they are responsible for needs to change, and if so make that change. From there it is a simple effort of going forward through the artifacts again to see if a change is required in the next artifact, and if that results in a change to the following artifact, and so on to the implementation itself. And again once that is done the result can be reviewed by going backwards through the process, eventually hoping to get all the way back to the Expert User with no issues (or at least no issues significant enough to require rework...).

 

# Back Page: HEMP *light* Artifact Flow Diagram