HEMP

(Holistic Enterprise Mechanization Process)

An agile approach to analysis and design

David E. Jones

Revision 1.0

http://www.DEJC.com/HEMP

ISBN-13: 978-1484184226

ISBN-10: 148418422X

Useful web resources

Author's consulting web site:

http://www.DEJC.com

Author's LinkedIn profile:

http://www.linkedin.com/in/jonesde

Author's original open source ERP project Apache OFBiz aka The Open For Business Project:

http://ofbiz.apache.org

Author's new open source projects Moqui Framework and Mantle Business Artifacts:

http://www.moqui.org

Author's GitHub page:

https://github.com/jonesde

The HiveMind PM application mentioned in the book:

https://github.com/jonesde/HiveMind

Preface

Audience

HEMP is for people who create software, especially software used to automate and manage business operations and general activities that involve multiple people and systems. This includes those who play the roles of expert user, business analyst, user interface designer, system architect, software developer, and quality assurance technician.

Some principles and practices apply to building hardware and machines, but are really meant for effectively handling the complex processes and activities that are the domain of modern enterprise software.

About the Author

I started my career doing software development, soon getting into packaged and custom enterprise software. I founded The Open For Business Project (now Apache OFBiz) in 2001. OFBiz is an open source business automation suite (eCommerce, ERP, CRM, MRP, etc) designed for easy customization and used primarily for custom internal use systems and as a foundation for commercial derivative works.

In 2010 I started a series of open source projects that are the next generation of the ideas and patterns introduced in OFBiz. These projects include Moqui Framework and Mantle Business Artifacts which together are a foundation for a variety of implicitly integrated open source and commercial enterprise automation software.

These free software projects are funded mostly through consulting work to customize and extend them. Along the way I have consulted on around 120 projects, including involvement with around 20 from beginning to end and participating in every step along the way. While I have a technical background, from the beginning of my career it was painfully transparent how often technical solutions were requested and unsuccessfully attempted for solving business and management problems.

My meanderings in the world of analysis and design were pushed along by various projects with a total lack of analysts and designers involved, and by unclear, incorrect, and frequently changing designs that never did what the business users needed and wanted. Sometimes even worse was a myopic vision of an organization constrained by an existing system, making a separation of requirements and designs impossible.

The result of inadequate requirements and designs is predictable frustration for both business users and developers, budget and schedule overruns, and even cancelled or abandoned projects. After participating in many such projects, when I started to take the helm I knew something had to be done differently and that

solutions from the traditional business analysis and software design world were complex and feckless.

The principles and practices described in this book are based on my consulting work, including a number of projects using HEMP itself. I originally put these ideas together and started using the term HEMP in 2007 while leading the analysis group in an OFBiz-focused consulting firm. HEMP has now contributed to the success of a number of custom software projects, and helped mitigate issues on others where it was applied either partially or late in the process.

About the Book

This book presents specific practices and tools for gathering and organizing requirements, producing designs based on requirements, developing software based on designs, and then making sure software implements the designs and the designs satisfy the requirements.

This book is a rewrite with improvements to HEMP as originally presented in the HEMP *light*, HEMP Complete, and HEMP Best Practices documents. While the general ideas are the same, recommendations about who should do what and how are slightly different.

Compared to the original documents, this book focuses more on business process stories and eliminates the recommendation to produce use case documents based on process stories (even on very large projects these have only limited utility). This book is also development tool and existing system agnostic so it does not mention OFBiz-specific

development artifacts or practices as earlier works did.

The term "artifact" is used in this book in its broadest sense as something created by a human as part of analysis, design, and implementation efforts. An artifact may be a document, diagram, or machine readable information such as code and data.

While there are various tools and artifacts presented in this book, the primary artifact is the **business process story**. This artifact documents the actors and actions of every business activity and structures them around the progression from one activity to the next. Business process stories are created primarily by business analysts working with expert users, and are ultimately used by everyone involved with creating and using the software.

HEMP is about high quality results. So many software projects, especially in enterprise software, devolve into mediocrity as the people involved struggle to respond to what seems like constantly changing suggestions and demands from stakeholders.

With good engagement early in the process and efficient artifacts to facilitate effective communication this can be avoided. More than that, the experience can be satisfying for all involved as the result takes shape through collaboration in a quality form more comprehensive than the best and most complete anyone involved anticipated.

Whatever your role in a project, I recommend reading through the entire book and then refer to relevant chapters the first few times you work on each artifact.

If you get bogged down along the way read Chapter 9 (Case Studies) for a little motivation and perspective.

The book is designed to be consumable in a couple of hours and to be easily referenced later on when you need a reminder of how to do particular things and what to keep in mind as you do.

1. The Same Thing Every Night

The title of this chapter is from a comedy sketch by Bill Cosby about recurring patterns in family life around bedtime with a handful of children. Whether little or big, people come across familiar situations and tend to behave in consistent ways when we do. With good practices beatings can be avoided, at least if there is no one on the project who is "always poppings peoples" (this joke has more meaning after watching this Bill Cosby sketch, available on YouTube and elsewhere).

When building software systems, especially large and complex ones, it might seem like an evening of parents persuading their children to peacefully settle in for the night. Amidst the chaos some things are the same every time.

Requirements versus designs

This is the most important principle of HEMP. Making a clear distinction between business requirements and system designs will make the biggest difference of anything you might do to improve success of software and other development projects.

- Business requirements in their most basic form are business activities that fit into a process.
- A business activity is an actor performing an action. It is described by specifying **who** the actor is and **what** the actor does to perform an action.
- The position of a business activity within a process shows **when** the action is done.
- Designs communicate ***how*** a user representing the actor will perform the action and ***where*** in the system it will be done.
- Neither requirements nor designs should try to answer the "why" question.

Requirements specify the **who**, **what**, and **when** whereas ***designs*** specify the ***how*** and ***where***.

When writing requirements it is common to discuss business strategy and the costs and benefits of a variety of approaches to achieve business objectives. Ultimately these discussions need to end with a decision on what will be done, and that decision is all that needs to be documented for building a system.

Others involved may want to document the rationale behind these decisions, but that should be kept separate from requirement documentation. Including the why in requirements makes them very large, difficult to create and maintain, and is distracting or even confusing to designers and developers who will be working from these documents. If you want to document the *why* use a separate *business case* document to keep your requirements and designs as simple and clear as possible.

It might seem difficult to distinguish a business requirement from a system design, especially if you

are mostly familiar with system designs labeled as requirements and have never seen a pure business requirement that is system agnostic (i.e. does not bias the design of the system).

Business requirements are about the business, not about the system. Designs are about the system.

When writing business requirements it is well worth the effort to use whatever verbal acrobatics might be required to avoid describing the system to be built.

For example, instead of writing:

"*When Customer submits order System sends Confirmation Email to Customer.*"

write:

"*When Customer submits order Company automatically notifies Customer.*"

Writing "*Company automatically*" keeps the focus on what is done, and leaves the how to the design. Note that the second sentence also leaves out the detail of how Company will notify Customer. This avoids biasing the design, but if all involved are sure that email will be the only way this is done that detail may be included without really crossing the line into design.

Another example of a common business activity that is tempting to express in system terms is when a user enters data or views data from the system. We know this will be done with a computer system, but in business requirements it is helpful to describe the

activity in a way that could lead to a pen and paper design just as well as a screen in an application.

Instead of writing:

"CSR types in Customer name, phone number, and address, and then submits the form."

simply write:

"CSR records Customer name, phone number, and address."

The principle to keep in mind is that business requirements should be limited to the business domain to make them an effective basis for design. The requirement information will then be flexible enough to facilitate creative and efficient design constrained only by actual needs of the organization.

Why bother with requirements?

The act of gathering and documenting business requirements in business terms helps engage users and sponsors of a project, and provides a clear direction to drive design and implementation efforts.

Engaging users and sponsors throughout the process improves alignment of the eventual system to the organization that will use it and sets the stage for satisfied acceptance, deployment, and long-term use of the new or improved system.

When working on designs without discussed and documented requirements different people are often thinking of different aspects of the business, sometimes different business activities altogether, and

often a different understanding of who should be doing what and when.

The typical experience of creating designs without requirements involves a number of people collaborating on design details with uncommunicated requirements in their heads.

The resulting designs are often poorly aligned with what the business needs. Getting to this false destination still involves long, frustrating discussions about details that seem irrelevant or nonsensical to others involved.

For business users driving the system design it is also easy to think a bad design is a good idea without considering who will be using it and what they will be trying to do.

Designing software is difficult and trying to design "cold" (without requirements) is a setup for failure.

Even end-users who are very familiar with business activities often fail in cold-design efforts. Without a discussion of requirements that set the context and clarify the details needed for design, it is difficult to remember everything and easy to get hung up on design elements that won't be used and may be expensive to build.

An initial focus on the business and documentation of business activities helps everyone involved to understand the business. Action-oriented business process stories provide a perspective on the business that is usually missed or glossed over in business plans and operating policy.

Writing process stories often involves uncovering details and prompting decisions that managers and executives have not addressed. The simple act of writing the story can tighten and clarify business operations, and involves a level of engagement and mutual understanding between those who run a business and those building software for it that is critical to the success and acceptance of the project.

In nonbusiness situations a similar pattern applies. Even consumer oriented software has sponsors and experts that drive the design.

Distinction between requirements and designs, and initial focus on business requirements is about building the right software and keeping everyone responsible engaged throughout the process.

What about "**analysis paralysis**" (endless cycles of analysis never leading to design or implementation) and "**failure to launch**" (project dies during analysis or design due to lack of confidence)? Effectively gathering, documenting, and organizing business requirements is the best way to get early traction in a project.

Quality requirements are immediately actionable for design efforts. Quality designs are immediately actionable for implementation.

If a project stalls during requirements gathering or design it means the practices for gathering and artifacts for documenting them are poorly matched to the task at hand.

This is a common problem because so many analysts are trained on dozens of diagrams and document structures that require significant effort but have minimal utility. By using HEMP and a focus on business process stories you'll have the opposite experience: active engagement and actionable results.

What about business users and sponsors who refuse to engage or aren't capable of discussing the details needed to build a system? These are other symptoms of ineffective requirements gathering and documenting, or of skipping requirements and diving straight into designs. Business users and sponsors are often alienated early on by poor communication and a focus on details that have nothing to do with their day-to-day activities. The focus on business process stories keeps the early discussions in a domain they are comfortable with, and allows them to provide actionable details.

The greatest resistance to HEMP usually comes from:

- business analysts trained in other tools and practices
- developers who have had bad experiences with ineffective and frequently changing designs

Business users and sponsors who have been through failed projects may have light initial resistance, but quickly appreciate the early engagement and effective communication facilitated by business process story writing.

Starting with requirements consisting of business activities is a way to begin with the end in mind and produce a system that will meet the needs of the organization.

Eventually a system needs to be delivered. When it is delivered it will be subjected to the ultimate test of alignment with business activities: actual use.

Planning for failure and change

Software design and development are creative human efforts that stem from human strengths and are sensitive to human weaknesses. Some weaknesses are part of natural human tendencies, others are cultural, and others are learned as a side effect of education and other life experience. Unavoidable human weaknesses lead to failures during analysis, design, and implementation efforts.

The more complex a project the more human weaknesses make things difficult. It is difficult for users to express what they need, and even remember and articulate everything they do. The leap between what they do (business activities) and what software might look like to help them do it is difficult, but at least manageable as long as what they do is effectively documented.

Because it is easy to forget activities and not realize nuanced dependencies between activities, let alone create effective designs for those activities, changes are guaranteed during analysis, design, and development processes. Any methodology for these activities must be sufficiently adaptable to accommodate and minimize the impact of change.

On top of human weakness, business environments and goals change over time and it is critical that the software systems an organization uses be able to

support changes within the organization and externally from partners, suppliers, and customers.

HEMP is a set of practices and artifacts for analysis and design that help keep changes earlier in the process where they require less effort to accommodate. This requires artifacts that are sufficient for capturing important details but simple enough for frequent change with minimal effort. The goal is the minimum effective set of artifacts, similar to the minimum effective dose principle in medicine. Beyond the minimum effective point come diminishing or negative returns.

This is the reason for the focus on simple artifacts like business process stories for documenting requirements, and avoiding artifacts like diagrams that are laborious to change during conversations and require supplementary documentation to be interpreted consistently by both business analysts and expert users who must create them and understand them to ensure they are correct and consistent.

Scaling complexity

Humans are not good at scaling complexity. We can only fit so much in our mental models of the world. As complexity increases the dependencies and interactions increase non-linearly, but even more significantly the human effort required to understand and automate the complexity goes up dramatically. The curve seems to be an approximation of x^2 so double complexity requires in four times the effort.

Necessary complexity can be managed by:

- refining and organizing requirements

- thorough design so implementation scales more linearly as complexity scales

Constraining complexity is easier to do during requirement analysis, especially when requirements are documented and organized according to business process.

Communication about business needs using requirements in the form of business activities makes it easier for expert users and other stakeholders to identify those that are more frequent and critical, and warrant more investment in automation. Communication about business needs using system designs makes it difficult for stakeholders to look at them in the context of the organization and evaluate their importance.

With requirements represented as business activities it is easy for stakeholders to understand the complexity they are requesting and give them a chance to simplify the business process or split activities into different phases to help prioritize and focus derivative efforts.

Trimming scope early (during requirements gathering) avoids unnecessary design and development effort which are expensive and time-consuming.

To effectively handle complexity in your project:

- write comprehensive and inclusive business process stories
- **review the stories and find ways to simplify and streamline business activities**
- make sure business process stories are easy to understand and well organized

- produce thorough, clear, and well organized system designs so they can be implemented literally
- review designs to make them as small and simple as possible while still meeting business requirements

Agile methodologies

Agile methodologies focus on development and rely on an external source for designs that drive the software development. A comprehensive agile approach including management, practices and tools adapts well to changing requirements and designs.

Agile methodologies prioritize frequent customer interaction and presenting results early and often. The biggest disconnect in the process is between requirements and design with a mix of both being presented to developers as a basis for their work. The customer is left on their own to make sure what they request of development is complete and will adequately satisfy organizational needs. The result is more difficulty for both customer and developer as they iterate toward what is hopefully a good result.

The solution is to help the customer by communication with expert users in language they are comfortable with, and that contains the details needed for design and development. With a focus on requirements the communication remains clear and to the point, providing a strong and flexible foundation for design and implementation.

The business process story and other HEMP artifacts can be used to drive a project that needs more predictive elements but the focus, as described in the

last section, is on adaptability and works well feeding designs to a development team using agile methods.

This is especially true for larger projects (such as ERP projects) where a minimum set of business activities must be supported before the software is of use to an organization. The temptation with agile development methods is to neglect analysis and design, even analysis as simple as documenting the business activities that need to be supported. The result is difficulty predicting the overall size of the project.

HEMP shifts the focus toward analysis and design while remaining adaptive.

To put HEMP in context it is similar to feature-driven development (FDD) with a focus on business activities and uses simplified artifacts to increase agility over the more complex and development oriented FDD. HEMP does not rely on object-oriented design and development and can be applied to a variety of architectures in existing systems and application development frameworks.

HEMP makes the most difference in automation of enterprise operations in any sort of organization where the business process involves interactions and hand-offs among a number of actors. Agile methods are best for projects involving incremental improvements but for these using HEMP will improve design and implementation system by basing them on requirements representing business activities and needs.

Diagrams

Humans have a natural instinct for language and even the verbally weakest of us can express and understand a wide variety of subtle ideas using language. Even so, it can be ambiguous and requires care in writing and review, with discussion among multiple people to be sufficiently clear and disambiguate subtle ideas. It also requires domain knowledge to anticipate common alternatives that people might consider and explicate which of them is desired.

Diagrams attempt to present a visual representation of ideas and disambiguate common ones with a variety of symbols. These symbol libraries are sometimes adequate for ideas of software structure and algorithms, but are rarely adequate for business ideas and the flow among the wide variety of activities necessary to operate organizations.

Diagrams are a sort of subset of language designed to represent specific ideas. They are low-resolution and without supporting text are often misunderstood.

Diagrams are cumbersome to maintain and keep consistent with supporting text and related artifacts, including other diagrams meant to represent different aspects of an idea or process.

The result is that diagrams are often not maintained during the high-paced changes that are natural when gathering requirements. On many projects where diagrams are used early on they are eventually abandoned while business process stories and other HEMP artifacts remain useful from analysis and design all the way through implementation and quality assurance.

2. The Story of HEMP

The Story of HEMP is a business process story just like the ones you will use to gather and organize requirements. It describes activities that make up HEMP in terms of the actors and actions. This story will give you an idea of what business process stories look like and introduce you to the business context that HEMP is designed for.

The bolded sentences are high-level activities made up of the more granular ones that follow. In large business process stories it is useful to create a summary story with just these high-level activities and links to the detailed stories that expand on them in separate documents.

Business Analyst and Expert User gather and document requirements. Expert User verbally describes business activities. Business Analyst documents activities in a business process story, asking questions as needed to clarify or expand on what Expert User described.

If Expert User describes a high-level idea and not a specific business activity then Business Analyst records it as an “Idea to Incorporate”. If the idea is one that cannot be incorporated into the story then

Business Analyst records it as a requirement statement.

Once the basic structure of the process story is in place, Business Analyst and Expert User review the ideas to incorporate and make changes to the story, modifying each relevant activity and adding activities as needed to represent the idea throughout the business process.

For critical system users Business Analyst optionally works with Expert User and actual users representing specific actors to write user experience stories. Business Analyst reviews user experience stories to ensure each activity is included in the business process story.

Expert User or other stakeholders optionally write a business case document describing general business objectives and their financial impact. Business Analyst and Expert User review business process story against the business case details to ensure business objectives are achieved by the documented business activities.

Expert User reviews the story and comments on incorrect or unclear wording, additional relevant details, and anything else that comes to mind while reading the document. Business Analyst revises the business process story until Expert User is satisfied that everything relevant is represented and Business Analyst is satisfied that the story is understandable and actionable.

Business Analyst documents overlaps and gaps with an existing system. If there is an existing system that will be modified or extended, Business Analyst

reviews each activity in the business process story and documents it as representing an overlap or gap in the existing system.

For each overlap Business Analyst documents how that activity is done in the existing system. For each gap Business Analyst documents any aspects of the existing system that are partial overlaps, and adds it to the list of gaps for design and implementation.

UI Designer designs screens and reports. Considering activities from the business process story and on gap descriptions (if available) UI Designer outlines the contents of screens and reports. UI Designer creates functional wireframes to accompany the screen and report outlines. UI Designer optionally creates screen flow diagram to show transitions between screens in each application.

Business Analyst reviews outlines and wireframes to verify the designs against the requirements. Business Analyst asks questions to UI Designer as needed, and may do the entire review in conversation with UI Designer.

UI Designer and Business Analyst review screen outlines and wireframes with Expert User by role playing. Expert User follows the business process story describing what they would do as each of the actors to perform each action. UI Designer plays the role of the system and describes how the system would respond to each user action, including changing from one wireframe to another as screens change.

UI Designer updates outlines and wireframes based on comments from Business Analyst and Expert User.

If comments from Expert User require business process changes Business Analyst updates relevant stories, and UI Designer updates design according to updated requirements.

System Architect models data and defines system interfaces. System Architect reviews each activity in business process stories and write data statements based on explicit or implied data to be recorded or reviewed by actors.

System Architect reviews screen and report outlines and maps each field to a field in the existing data model. If there is no adequate field in the existing data model, System Architect records data statements describing the field and its relationship to other data concepts.

System Architect reviews user interfaces and based on anticipated system architecture (especially for client applications on mobile or desktop devices) designs services and/or API for processing user input and preparing more complex data for presentation.

System Architect reviews business process story to identify all system-system interactions (as opposed to user-system interactions) and identifies an existing system interface for each, or defines a system interface (web service, file drop, API call, etc). System Architect maps each field in system interfaces to the data model. For fields without an existing field in the data model, System Architect records data statements describing the field and its relationship to other data concepts.

System Architect organizes data statements by data concept to group them for easier modeling and to remove redundant statements. System Architect maps each data statement to the data model and as needed extends data model based on data statements. System Architect updates user and system interface data mappings for relevant fields.

System Architect defines initial and test data to demonstrate how data is structured and to use for testing.

Software Developer implements user and system interface designs. Software Developer reviews business process story to understand the context of what needs to be built. Software Developer implements software described in user interface designs and technical designs.

System Architect, UI Designer, Business Analyst, and Expert User review implementation. System Architect reviews implementation to ensure that data comes from and goes to the fields described in data mapping. System Architect reviews service and/or API implementations and other system interfaces for consistency with the technical designs.

UI Designer reviews user interfaces and tests interactions against descriptions in the screen and report outlines, and layout against the wireframes.

Business Analyst reviews implementation by performing business activities described in the business process story. Business Analyst hands off each activity that is functionally supported to the Expert User for final review and testing.

If QA Technician is involved, QA Technician performs a comprehensive test of implementation against individual designs and end-to-end test based on the requirements in the business process story. QA Technician identifies and tests possible uses of the implementation that are not part of the business process story, and not an explicit part of the user and system interface designs.

With comprehensive testing done by QA Technician, other roles can reduce their efforts to spot review and testing, except for the Expert User who should review everything from a user perspective for acceptance of delivery.

This completes the story of HEMP. The activities described here tell you what to do (the requirements of HEMP), now we'll go over how to do them (the design of HEMP).