HEMP Best (and Worst) Practices

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

Overview

My background is in open source enterprise automation software. There are excellent open source code bases available for every sort of application infrastructure and even ready to use enterprise automation components. These open source commodity software elements, and certain commercial offerings as well, give developers more flexibility and efficiency than has ever been possible before. Business and personal objectives can be automated with so much ease that efforts more and more tend to succeed or fail in spectacular ways.

Most projects succeed or fail before development begins, or because of factors that have little to do with development. Those factors are usually not recognized until late in the development process where the rubber really hits the road. With so much power and so much variability some understanding of good analysis and design practices for development purposes, and related common missteps, is invaluable.

These books will cover recommended artifacts and processes to facilitate communication and group efforts to move from gathering and formalizing general concepts and ideas (requirements) all the way through implementation and testing using a business process focused approach. In other words this looks at the “whole” process of activities related to building enterprise automation (mechanization) systems, hence the term “holistic”.

The approach presented is a complete process and set of artifacts that ties together many common tools in a consistent and easy to follow way, even for large groups. The artifacts include common tools such as use cases and stories that are often misapplied or used ineffectively because of a lack of definition and understanding of what those artifacts are for and how they will be used in subsequent artifacts.

These books includes descriptions and examples of various analysis artifacts including stories, statements, and business process use-cases for requirements gathering, and system interaction use cases, screen/report outline, and data statements for initial design. Artifacts that would typically flow out of those initial design artifacts include things like screen/report wireframes, relational data model definitions, automated process specifications, and mapping between them as applicable.

These designs can then be more or less literally translated into a system implementation. Implementation factors will be discussed, but actual implementation processes are intentionally out of scope for these books in order to make it as technology-neutral as possible. Certain testing artifacts are included here, though actually running the tests (manual or automated) is a part of implementation efforts.

It is not the intent of the HEMP books to cover a wide variety of document or diagram structures that can be used for requirements gather, analysis and design. There are dozens of books that cover a wide variety of these, including pretty much everything in these books. There are even some books which cover a lot of these, like Requirements Analysis: From Business Views to Architecture by David C. Hay.

HEMP will also cover how certain artifacts and steps can be skipped for efforts with fewer requirements and fewer people, with a focus on keeping the ones that make the biggest difference.

Parts of these books will include discussion of the transition from design to implementation in terms of the Apache Open For Business Project (OFBiz), but all analysis and design artifacts discussed outside of implementation are applicable to any technology or set of systems.

Why “HEMP”?

The idea behind HEMP is to have a standard way to prepare for enterprise automation efforts that considers a complete view of activities in an organization. The words “Holistic Enterprise Automation Process” accurately describes this, but HEAP is somewhat disappointing as an acronym because of various possible interpretations, and is no good for line art either. The word “Mechanization” is a nice synonym for “Automation”, and makes a rather nice acronym with interesting, if not generally understood double-meaning: HEMP.

Hemp is unfortunately better known for its cousin plant, but itself is in fact a useful plant without mind-altering side effects (which is not what you want when preparing for enterprise mechanization!). It is known for strong fibers (used paper, clothing, rope, etc), and healthy food (good protein, balance of omega 3 and 6 fatty acids, fiber and other nutrients). Also, interestingly, it is illegal to grow hemp in the USA but it is grown in nearly every other country around the world.

Event the best analysis, design, and implementation practices are easily corrupted, and just like HEMP the plant hemp can be contaminated by fungal and yeast growths, so you'll see many parts of these books warning of pitfalls and recommending practices that will give you the best chance “out there”.

Influences and Background

Some Interesting and Helpful Books

Robert M. Pirsig, Zen and the Art of Motorcycle Maintenance and Lila

Donald Norman, The Design of Everyday Things and The Design of Future Things

Eliyahu M. Goldratt, The Goal (with Jeff Cox) and Beyond The Goal

Alistair Cockburn, Writing Effective Use Cases, others by Cockburn

Frederick P. Brooks, The Mythical Man-Month

Graduate Work on User Interface Design

During my short time in graduate school I studied human interfaces, with an interest in human augmentation in general. One of the more interesting aspects of human augmentation is to enhance what groups of people can do working together, as opposed to much of the field which is dedicated to enhancing what an individual can do. Enterprise systems in general terms are just tools to enhance what groups of people can do, especially related to coordination and management of efforts.

Many concepts used in human interface design are applicable for organizations as well. One of the more important is perhaps end-user testing to make sure that proposed solutions are applicable and effective. This can be done based on designs to test proposed screens, and can also be used to validate stories in a role playing sense to make sure hand-off between actors is handled well and that there are no missing steps in the written documents, among other possible shortcomings.

Good and Bad Consulting Experiences, Observations

In my professional life I've attended training seminars, read dozens of books, and used dozens of different software tools that are all meant to help with analysis, design and eventual implementation of software and other types of systems too. Being a creative effort there are no deterministic processes to follow and adaptation seems to always be necessary. However, there are certain tools that effectively address certain needs, and that if done right transition well into other tools that address more needs closer and closer to implementation.

I've personally tried many different tools in my 10 years of designing and building software systems. I have also had the privilege of working with well over one hundred clients and hundreds of capable professionals around the world. Some have used tools and applied concepts in downright inspirational ways. Others have struggled to even identify what can and should have an impact on the eventual system and what in the system can should have an impact in the organization. It probably goes without saying that most people are somewhere between, but what is more difficult for all of us is to see what will result from our efforts and what and how we can do things to lead to more consistent and beneficial results.

Of all practices, tools, and concepts the most important seems to be the ability to distinguish between requirements and designs, and to make sure to start with requirements, and then create designs based on the requirements. A similar concept in the natural world is identifying cause and effect, and that can be illusive as well once you really start studying (David Bohm's book “Causality and Chance in Modern Physics” has an excellent discussion of this). That is of course one little part of this topic and many others are presented herein, including stepping beyond such distinctions and into trying to optimize for “quality”, which while useful is another somewhat illusive concept (which is well covered in general terms in Robert Pirsig's book “Zen and the Art of Motorcycle Maintenance”).

The reason for introducing such concepts, and applying them to real-world experience, is that we can only hope to improve results in future efforts by noticing patterns and principles from past efforts and using them in the future. Some patterns and concepts end up being useful, and others don't seem to make any difference. HEMP is not just a description of tools to use, but reasons why and the more general objectives they help you reach as well as the recommended route to those objectives.

Note on Empirical Versus Statistical Basis

The ideas in these books are primarily based on empirical observations, or in other words inducing principles based on details (inductive logic), and actual things to do can be derived from these principles and recommendations using deductive logic. Still, however good your logic may be watch out for what is effective and what is not. Ask “how is this working?” on a regular basis and adjust as needed to hit the most important things and get done what is needed.

Chutes, Ladders, and Pitfalls

Chutes and Ladders can be fun. This is especially true if you enjoy watching children try to attempt more “control” once they’ve realized that with a 100% luck-based game there isn’t much they can do to influence whether they win or lose. With analysis and design there is fortunately not as much luck involved, but there are numerous pitfalls. Pitfalls are categorically not fun.

In various parts of the HEMP *light* and HEMP Complete books there are sections about common pitfalls related to the artifact or effort being discussed. There are more general pitfalls to watch out for as well and these are presented in this book. If you want a good overview of why to use HEMP and the general rationale and motivation behind it, this is the book for you.

How Does HEMP Relate to Agile Processes?

Agile processes are excellent for smaller applications and for maintenance and enhancement of larger applications. If the application is larger and needs to be deployed all at once the less formal design and development (often without requirements gathering) combined with smaller iterations such as the common two week iteration, the result is often delayed project completion (and inaccurate estimates of completion dates and resources needed), and if decision makers doing the iterative designs do not collaborate enough with end-users the result may not be useful to the business. More likely certain parts of it will be usable, but it will have to be used along with other applications and manual processes to fill in the gaps.

There are many ways to work around these typical weaknesses in agile processes, and using certain parts of HEMP is one way to do just that. HEMP *light* is actually meant to be used to support a more agile project, with tools to define the big picture with the Process Story and with that keep track of everything.

In the HEMP *light* book it talks about how it is meant to be used as a lighter weight process where there are fewer people involved and the overall business process is not too large and complex. It leaves out a number of the more formal artifacts while still keeping what is needed to define the big picture and dive into each part of that picture for design and development.

Combining HEMP *light* with an agile development process is an ideal way to go for both initial implementation and ongoing maintenance and enhancements. For ongoing work the HEMP *light* artifacts can be modified, starting as early as needed (often with the Process Story), and then making changes in following artifacts based on those changes all the way through the development of the system.

For larger projects and where more people are involved agile development processes can be used once the design is complete for each part of the system. In fact, agile development still usually works quite well at this stage. What doesn't work so well for large systems where many people are involved is doing iterative requirements gathering and design. In this case doing analysis and design iteratively will result in losing the big picture and major gaps in the end result that must be filled with design and implementation changes, or with manual process and external systems that make things significantly more complicated for end-users.

How Does HEMP Relate to Object-Oriented Analysis and Design?

Many of the activities and intents of HEMP are similar to traditional object-oriented analysis and design. By leaving out object-oriented patterns and in fact leaving out objects altogether, the analysis and design efforts are both more simple and more relevant to the business requirements. The artifacts produced are focused on what really matters in the applications from a business perspective and are sufficiently detailed to result in an application that is exactly what was expected, and also still technology neutral enough to allow for implementation using object-oriented or any other technology approach.

Just in case I haven’t made it clear yet, it is the author’s opinion that object-oriented design and implementations patterns are the worst thing that has happened to enterprise automation software in the last 30 years. OO results in large and cumbersome code bases that are generally difficult to create and maintain and that introduce concepts and relationships that are artificial and don't match what organizations deal with from a real world business systems perspective. Because nothing is naturally OO in the business world everything must be mapped to objects so you get an explosion of objects and code to map everything to objects including database (or general data store) structures, file formats, system-to-system message formats (even for SOAP, XMLRPC, and plain-old-custom-XML), screen and form elements, user and system events and handling them, and so on.

None of these things is naturally expressed as an object so code must be generated or manually written to represent each of these things and significantly more code must be generated or manually written to implement the dozens of mappings between them for each aspect of the system. It is my opinion that any time you must use a code generation tool it means the language being used is not well suited to the task at hand. Generated code is easier to write, but is still cumbersome and inflexible and therefore difficult to maintain and refactor as requirements and designs are expanded and changed.

How Does HEMP Relate to the “Best-Of-Breed” Approach to Systems (Integrated OOTB/OTS Systems)?

The goal of HEMP is to enable what is almost the opposite of the best-of-breed approach. This is an unfortunately common approach: take OOTB (Out Of The Box) or OTS (Off The Shelf) software and try to integrate many applications together to make them more applicable to the organization, and then supplement it with informal “systems” such as spreadsheets.

When doing things this way initial requirements are often high level and vague (if defined at all), and need to be so in order for it to be possible that an existing system will be able to meet the needs. During implementation the business is forced to make hard decisions about doing things the way the software supports or building around it with formal or informal systems, and it is an uncomfortable process with constant compromise of the desired business processes. The general danger is always that if you don’t establish the right questions (“requirements”) up front you’ll get an answer (“design”) to the wrong questions and it will have limited use in your organization. With the best-of-breed approach the answers are already there and you don't get a chance to ask the right questions, and the result is a system that never really quite fits.

The HEMP approach is to gather requirements in detail and then design based on them, and then build or customize a system to match those designs. This is best done with systems that lend themselves well to gap analysis and reuse which is an improvement over building from lower level and more technically oriented tools. This can be applied to systems that are more difficult to customize and that are easier to use in the OOTB/OTS way, but you will find more compromises being made and the greatest advantage will be that your detailed designs document where the systems fall short of what is desired and the compromises or alternatives decided on. It will be hard to fix the problems, but at least you'll know what they are. In some organizations that may not be a benefit and if management has decided to go with the OOTB-/OTS-based best-of-breed approach they may have good reason for not wanting to use something like HEMP.

While it is called “best of breed” the real fact of this approach is that when going from one system to another in the middle of business processes you can only use features in one system that are supported by the next, resulting in a “least common denominator” reality where “best of breed” was desired. In some cases you are forced into this circumstance because of existing systems you are not allowed to replace. A lot of technology and practices have been developed to support the best-of-breed (ie least-common-denominator) approach, but that doesn't mean it is a good idea, it's just something people get stuck with sometimes for the legacy reasons I mentioned above. If that isn't the case for you then don’t choose to get into that uncomfortable and always less than ideal situation when you don't absolutely have to.

What About Being Flexible and Supporting Changes?

The artifacts and process described here may seem cumbersome (especially for HEMP Complete), but actually having a progression of artifacts from less to more formal means that more changes and decisions are pushed to early in the process where changes are easy and cheap.

Even late in the process where artifacts are more formal and expensive it is invaluable to be able to go back to less formal artifacts (as far back as the change has an impact on an artifact), make changes there, and then work forward from there doing a number of easy things with little risk instead of one big difficult thing where a mistake requiring further rework is likely.

Level Of Effort for Analysis and Design

There are two primary factors that influence the overall level of effort for analysis and design projects: the number of people involved in specifying requirements, and the scope of business processes and related requirements that will drive the design of the system that will automate those processes and requirements. While there are certainly other factors these two are the most important and have the greatest impact on the level of effort and cost of analysis and design.

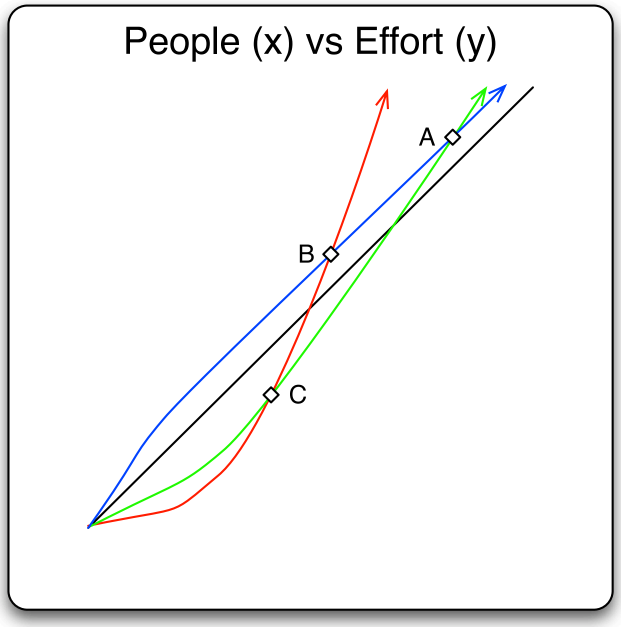
The diagrams below include one diagram that shows the relationship between the number of people involved and the level of effort, and one diagram that shows the relationship between the scope of the requirements and the level of effort. In both diagrams you can see that different practices results in a different level of effort for a given input of the number of people or scope of requirements.

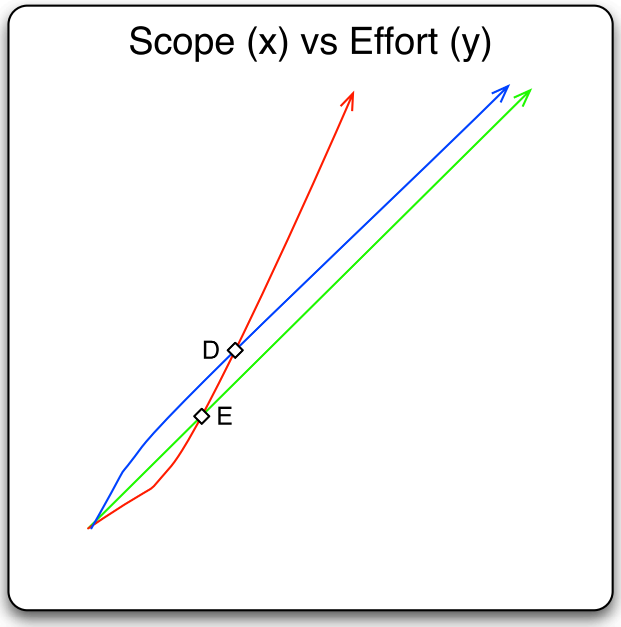
In the first diagram there is a black baseline and a red line that represents either no process or a very light process. This line scales slowly for a small number of people and at a certain point starts to increase quickly so that adding more people results in a substantial increase in the overall effort. The blue line represents a heavy process and scales quickly at first but then more slowly over time and is eventually far more efficient than having no process or a lighter process. The point "B" in the diagram is the point where a heavy process becomes more efficient than a white process. The green line represents HEMP, which lives somewhere in between the light and heavy processes. For very small groups HEMP may actually be less efficient than no process or a lighter process, and that is the area before point "C" in the diagram. Beyond that point HEMP is far more efficient than no process or a wider process. Even the HEMP process only scales so well and that a certain point will be less efficient than a heavy process. This is represented by point "A" in the diagram. Note that no matter how good a process is it will never scale linearly. In other words, regardless of the process at some point adding people will result in diminishing returns.

There is a similar pattern for scope versus effort. One major difference with scope is that there are ways of handling it such that an increase in scope results in a linear increase in effort, and that is what HEMP is all about. For a very small scope having no process or a very light process will actually be more efficient than using HEMP, even HEMP light. This is represented in the second diagram as point "E." Once you get beyond that point HEMP is significantly more efficient. Unlike with the number of people when considering scope HEMP is always more efficient than a heavier process. Heavy processes scale poorly at first and then scale close to linearly but always with a certain overhead that keeps it above the optimal line. In the diagram this is represented by the blue line for heavy processes staying above the green line for HEMP.

In addition to these general trends sometimes there are things you can do to help flatten the curve. For example with the number of people curve you can reduce the size of the people that effectively specify requirements by selecting a small group to represent the larger group. This smaller group would collaborate with the larger group but ultimately make the decisions about which requirements will be used.

It is important to keep the estimated level of effort as low as possible and use practices that help the level of effort to scale more reasonably as the number of people and the scope of the requirements increase. Keeping the level of effort estimates as low as possible will also help you focus on the most important aspects of your project.





Base Line
No/Light Process HEMP Heavy Process

Where to Put This Stuff

By nature the information in these artifacts is created and used by a number of different people. Because it is impossible to get the best results in a first pass, supporting ongoing changes while working against a certain state of prior artifacts is important. In short some sort of system is needed to help the people involved (end/expert-users, analysts, developers, and others) collaborate and do the work they need to.

The general process for handling the documents and hand-off to other teams, or for use in subsequent artifacts, is to select a specific revision for each relevant artifacts to be used in the next artifact and then work from that revision for each document, allowing others to change the documents resulting in new revisions that don't affect the work being done until it is decided to work from a new revision. When choosing to work from a new revision rather than reviewing the entire new document those involved with follow-on artifacts can just look at the differences between the new revision number and the old one. With larger teams and the need for feedback from different people it is nice for people to “edit” the document and leave comments and changes in-line, and then save their changes as a variation to the document so that it is easy to see what they changed and the document owner (the person working on the document) can then incorporate those changes selectively.

For larger systems the requirements tend to be somewhat large, and because there is a progression of artifacts from less formal to more (all the way through to the code), there will be many artifacts that are linked to other artifacts. For a given artifact it is valuable to see incoming as well as outgoing links, and because of the difficulty in keeping lists of incoming links manually it is best if the tool can do that automatically. It is also common to have more than one artifact in a single page (to make them less cumbersome to work with), so linking to another artifact directly means being able to link to a section within a page (or document) rather than just to the entire document.

Based on these the general requirements include:

- Revision management
    - history of changes
    - diff between any 2 revision numbers
    - preferably revision # for a whole set of pages, but at least for each page
    - the ability to “branch” saved changes to create a variation on the page
- Hierarchy of pages
- Sections within pages
    - tied to headings
    - hierarchical headings (ie h1, h2, etc)
- Links to pages, and sections in pages
    - alternatively use no links, maybe some indexes at most, and use consistent terms and searching to manage relationships

There are various tools that meet these requirements for text documents, and a lightweight formatting, like a wiki format of some sort, to help keep documents consistent is really valuable. A nice commercial one is Confluence (an “enterprise wiki”) from Atlassian. There are various open source wiki systems like twiki that may also work well for you.

One thing that is difficult to work with in a way described by these requirements is graphical files, including diagrams. Along with other reasons that is one reason why there are not very many diagrams recommended in these books. Still, diagrams are vital for representing visual concepts and spatial layouts.

When diagrams and other visual documents are needed they can usually go in the same collaborative content system as text documents, but the revision control is more limited (no incremental differences, just complete copies of each revision of the document) and getting a view of differences between revisions is rare and not generally available.

The most popular diagramming tool is probably Visio on Windows, and that is certainly adequate for any diagram or wireframe described herein. On the Mac a great tool for diagramming is OmniGraffle, and that is the one I most commonly use. There are some open source tools, like Dia or Kivio or even OpenOffice Draw, and those should be adequate for pretty much any of these artifacts even though the tools are generally more simple and less feature rich that Visio or OmniGraffle. Many UML diagramming tools are also adequate for some of these, but they tend to be more limited for general diagramming and more oriented to coding and things like round-tripping between code and the diagrams.

The Same Thing Happens Every Night

The title of this section comes from a comedy bit from Bill Cosby that covers some common elements family life, especially around bed time with a handful of children. Whether people are little or big they face similar situations all the time and tend to behave in similar ways. When building systems, especially large systems, it often seems the same as an evening with parents and a handful of children preparing for bed. The same things tend to happen every time.

Requirements versus Designs

In life the love of money may be the root of all evil (Radix malorum est cupiditas), but in software projects the root of all bad results is Designs without Requirements. Love of money can certainly cause problems as well... but let's get back to the issue at hand.

Before getting into this I should qualify this problem as mostly applying when there is more than one person specifying requirements, or more than one end-user who has a say in how the system will work. If you have a small scope for a single user, gathering and formalizing requirements before doing designs may be more work than it's worth. See the notes on Small Teams below for this.

Some common misconceptions and less effective approaches:

- Requirements = Designs
- Designs instead of Requirements
- Designs then Requirements (ie Requirements derived from Designs)

Some main results of requirements gathering done correctly:

1. If you have more than one person with input on what the system should eventually do, then gathering and formalizing requirements helps to get everyone on the same page. That is perhaps the most important result of the effort.
2. A close second place result is that ideas about what needs to be done and how will arise during this process that would not without creating and reviewing non-design artifacts that are focused on requirements.
3. Third after only those two is that with requirements developers will have a source for answers to questions that may not be addressed in the eventual designs. Without knowing where a design idea came from it is easy to misunderstand the design and implement incorrectly.

If you've ever done a post-mortem on a failed project you'll probably recognize elements of this list. These are the most common things that cause problems later in a project, and are often the ones that lead to ultimate failure of the project.

To put it bluntly, while gathering and formalizing requirements before doing designs will not guarantee the project will be a success, not doing so is a very good way to increase the chances of project failure. This usually looks like a system that has to be changed so much in order to meet business needs that if it ever does go into production and is used by the business, it will only be temporary until a replacement is ready.

Requirements and designs are separate and complimentary concepts. To demonstrate the concepts here are some of those fun SAT-style analogies:

- Questions : Requirements :: Answers : Designs (in terms of questions leading to answers, questions being the reason and guidance for finding the answers)
- What, When : Requirements :: How, Where : Designs
- Business Activities : Requirements :: System Interactions : Designs

With the stories we want to stay clearly on the side of requirements and defer designs until later. Related to the first item: if we don't know the question, how can we find the answer? The point of the Process Stories and Requirement Statements is to gather the information unbiased by a design so that we can make a good design.

In general for Process Stories any mention of the system to be built should wait until later, in the system interactions. Those will be in Use Cases if they will be done, or in a separate document, or even just in the screen outline if the effort is simple enough to go right from stories to screens (outline and wireframes).

Here is an example of a story fragment that is too biased towards design and doesn't really mention requirements: Customer click "Contact Us" and then fills out a form that drop dynamically over the current screen. The user enters a subject, message and (email address if not signed in, otherwise email is listed in the "From:") and submits.

That is a system interaction description, and not a business activity at all. It is a design and not a requirement. One of the reason to start with requirements is that it is hard to work backwards as designs bias requirements so much, ie there is no room or reason for discussing the "question" that lead to the "answer" because you already have the "answer". Sometimes in order to check the "answer" with a "question" you have to first throw away the "answer".

One possible business activity sentence that might lead to this design is: "Customer contacts Company electronically, specifying a subject, message, and if not already authenticated then also their email address". To put that into context there should be things before it that describe what gets the customer to that point, or perhaps just a prefix to set the stage of "If Customer has a problem, needs an answer to a question, or...". Doing that would help with the design since they know what the point of the it all is. Following it you'd expect to see what naturally comes next in the story, ie what does Company do when they get the message, etc?

Following this pattern will help you build a solid foundation for the designs and eventual system, and help make sure the solution is applicable to the business and meets its needs, and that resolves the most significant and common cause of project failures.

Planning for Failure

The only thing really consistent about human performance is failure. The most successful people and groups start out by acknowledging for themselves and others they work with that they are human and will frequently and consistently make mistakes. Any process or tool that does not include a plan for finding and dealing with mistakes is not likely to contribute to the overall success of a project.

The rather painful lack of omniscience in human nature is why things will change over and over, and over and over. All processes and tools need to make these changes as cheap and easy as possible. While humans aren't very good at knowing everything in advance, they are very good at noticing problems later on, sometimes at awkward times such as the middle of the night or in the shower while singing show tunes loud enough for the neighbors to hear. Sometimes it takes a while for even the best brains to process a huge volume of information, even days and weeks, but eventually the stuff tends to surface as the people involved get into things more and have more chances to think about them.

The Mythical Super-Man-Week

The Mythical Man-Month by Frederick P. Brooks includes essays that describe a number of good principles related to software development, and a few that relate to analysis and design, though mostly from a technical perspective. The essay that the title refers to describes how scaling a team to reduce timelines will result in non-linear and inefficient improvements (diminishing ROI), and may actually result in extending a timeline instead of reducing it (negative ROI). This makes a lot os sense in simple terms. Time spent represents cost, not value created for the project or organization. If any effort is not focused on creating value in a coordinated way there will be costs but may be no value created.

That is one of the many resource and estimation related pitfalls that inexperienced managers and project sponsors will run into. Another related, and interesting and important one, is that as complexity increases the overall effort involved does not increase linearly, it increases something closer to exponentially. For example if you double the scope or complexity of a project the effort required to complete it will likely quadruple and if you triple the scope the effort required will increase nine-fold. There are many reasons for this, not the least of which is that the bigger the scope and greater the complexity the more people will likely be involved and with more people and more complexity together the percentage of the overall effort dedicated to coordination and communication goes up significantly.

In really large projects it is common for an analyst or designer to spend over four hours on communication and coordination for each hour they spend producing an artifact. Even worse without a lot of focus and clear direction followed by peer-review many of the artifacts produced end up getting thrown away and either rewritten, or restructured and fleshed out, or entirely not used because the artifact was not adequate for the need or the direction turned out to be wrong.

In other words, my point is definitely not that as a manager or team member you should push for less coordination and communication, but rather that you should plan for it and be realistic about it. In planning for it and being realistic about it if you have strong time and/or experienced resource constraints, you should address this issue by reducing scope and complexity, not by adding people or trying to skip important efforts and artifacts.

Another tempting solution to a time problem is the Super-Man-Week. Everyone has experienced this to some extent or another. Many of us have pushed others to do this. Many of us have been pushed to do this. Many of us have pushed ourselves to do this. In some cases managers even consider this as a solution to budget problems as well as time problems because they sometimes get away with not paying salaried employees for additional work hours in a week, and also not increasing future time off based on the upfront additional time. These cases go beyond the inherent problems of even well compensated long work weeks because of the problems created with working relationships and personnel morale.

Attrition will always increase and those who want to leave but either can't or are afraid to will not work very well. In spite of financial and planning figures appearing to look better this is always a lose-lose situation, or in other words both the organization and the individuals see all of the important factors reduced, especially quality, efficiency, productivity, satisfaction, and sustainability. Both organizations and individuals are affected by all of these, and both organizations and individuals can experience burn-out which results in a sustained decline for all of these factors.

For anyone who cares about the success or failure of a project those aren’t even the most significant impacts on a project. In fact, unless the project is being carefully managed by experienced people you probably won’t even noticed the effect of long work weeks that results in the greatest impact on time and budget. It’s all about priorities and focus.

The most important thing for a project to succeed is to focus on the "critical path" and the most important and solution affecting efforts. By focusing on a solution of more hours per week you introduce a lot of hours where people work inefficiently and are more likely to make bad decisions about requirements and designs, which will have a huge impact later on in the project. They are also more likely to not focus on the highest priorities and the efforts and artifacts and details that will have the highest impact on the effectiveness of the solution for the business goals it is meant to help automate or optimize. Any time spent on something that is a lower priority puts higher priority efforts and artifacts at risk.

There is a big discipline aspect to this. When people know they will work long hours in a day and they are feeling stressed and tired they will work on the things they want to work on instead of the things they know are really important and that must be done. This leads to more and more wasted time and may actually result in the higher priority efforts and artifacts getting less attention than they might during a shorter work week with more discipline and regular review of priorities.

The focus on establishing and working by priorities is the solution to this. People need to feel empowered to work together and decide what are the most important things to be working on, and expect management to work with them and support them in those decisions, understanding the not everything conceivable will get done. The real strategic objectives and project success evaluation criteria need to be defined better as things seem to be getting off track.

Strategic objectives are the aspects of business strategy that relate to the project, such as:

•introduce new offering XYZ

•improve customer service and repeat sales by...

•increase production throughput (measured in terms of actual sales)

Project success evaluation criteria are things to be reduced or increased. Sometimes these can be measured quantitatively, but sometimes they can’t and all you can do is look at whether a certain action is likely to increase or decrease that criteria. Common examples include (along with desired direction):

•project/solution cost (reduce)

•time to delivery (reduce)

•quality of solution (increase)

•value to the end-user organization(s) (increase)

According to the objectives and success criteria team members can work with managers and sponsors to get the important stuff done. This does mean that managers and sponsors must be willing establish and respect priorities, and not just accept it when lower priorities are never touched, but also encourage people and appreciate it when the higher priorities are addressed. As long that the priorities are established well (easy to do with objectives and success evaluation criteria defined), this is a win-win situation for the organization and all individuals involved.

One example of these criteria and priorities in action related to HEMP is doing things out of order or when dependent artifacts are not complete. What will happen if you start creating the Data Model before the Data Statements, System Interaction Use Cases, and even the Process Stories are done? If you are using a simplified process you may choose to skip Data Statements, and possibly even skip Use Cases and just write System Interactions based on the Process Stories, but the point is what will happen if you start the Data Model before whatever you want to base it on is finished? For the criteria mentioned above the likely impact is:

- increased project cost due to a high chance of work done not being used
- if changes are not tracked carefully the quality of solution will go down
- if assumptions made during this preemptive data modeling are not discovered and cleaned up the value to the end-user will be reduced
- if there are not significant changes in the dependent artifacts after you start you may reduce the time to delivery, but if there are significant changes you may actually increase it

Based on this it is clear that this is a bad idea because the only real possible upside is reducing time to delivery, but there is only a chance of that and in fact there is a chance you will actually increase the time to delivery in addition all of the other criteria being pushed in an undesirable direction.

So what is the take-away from all of this? How do we increase our chances of success when there is too much to do and not enough time and people to do it?

1. For a fixed time period instead of increasing people reduce scope and complexity.
2. If you increase scope and complexity plan for an exponential increase in time and budget.
3. For a fixed budget reduce scope and increase planned time to delivery.
4. If interim deliverables are coming in late and timelines seem threatened establish priorities and focus on them instead of asking people to work longer weeks or to cut corners.

There is the saying about working smart instead of working hard, and if you can get away with that then great! Still, don’t get your expectations up. Make working smart a priority and work hard only if you have to, and only once you’ve established the way to work smart so that you’re headed in a good direction. Consider working hard like speed and working smart like direction. If you’re going fast in the wrong direction you won’t end up where you want to be, and in fact you may end up very far from there. In other words focus on priorities and impact, not on long hours.

Small Teams, Lightweight Process, and Refactoring

The ideal circumstance when developing a system is to have a small scope to address (or a series of small, semi-independent scopes...) and a small team of very intelligent people to work on solving the problems within that scope. Projects like this tend to succeed in spectacular ways. When you have a half-dozen people including the users and developers working together closely (daily interactions, preferably in the same office space, etc) it is easy to solve problems that come up and do things "in your head".

When working on larger projects and/or with larger groups doing things in your head and trying to start moving without establishing some direction can cause significant problems. Even with the situation where you have many users instead of just one things break down quickly when you try to create designs without requirements or change things on the fly, or more specifically, change things later in the design and development process.

In terms of resources and management what this all comes down to is that the more people involved and the bigger the system the more expensive it becomes to change things later in the process. This goes back to the Requirements versus Design and Planning for Failure patterns. When you know things will change, and that it becomes more expensive to change things as artifacts become more formal, it is better to communicate in the open with artifacts that are informal at first and more formal later so that things can be recognized and aligned as early on as possible.

Small teams using a very lightweight process and using refactoring instead of heavier upfront analysis and design is wonderful and by far the best approach for certain efforts. One common example of this is ongoing maintenance and incremental feature additions to an existing system. Where efforts are small and adaptive by nature the most important things are developers and users working closely together and a very flexible system and toolset so that changes can be made without too much difficult or expense or compromise in design.

Academic Understanding with no Experience or Mentoring

Analysis and design of large systems are complex and involved efforts that call for a lot of experience with whatever the system is meant to help automate, and a lot of experience gathering and formalizing requirements, and then turning those into designs that can be acted upon in a reliable way.

For analysis and design efforts in general there are a staggering number of tools available and proponed by one expert or another, and deciding which are best to use and how they will fit together or flow from one to the next can be very daunting, and risk and error prone as well.

Especially for larger projects with many people involved (ie where coordination is required and consistency and organization become huge success factors), it is important to have someone with experience to help out. That person should either drive the process altogether, or at least help define the artifacts and processes to use, and then work with everyone on what they are doing and regularly review what is being produced to help keep things focused. The goal is to help others get traction and make progress in the direction that has been decided on, and to continue heading there in a reliable way.

Establishing but not Understanding, or Understanding but not Following

This is important for both process and artifacts created during a process, and becomes more critical for larger team and project sizes. The most common symptom of not effectively following good processes is that people are finding their assignments to be very difficult, or they are confused about what to produce and what to base it on and how to move from one to other. In order to solve the problem it is important to understand what is causing people to not follow the process.

It is easy to establish a process and artifacts, and still suffer from not everyone involved understanding the process. This generally results in those who are not understanding performing poorly, missing deadlines, and generally having a hard time getting things established and organized in their area, and making progress toward the various deliverables that are defined. One of the great dangers of this is that those who don't understand may not realize it, which aggravates the problem because everything is perceived as being normal and going well or as expected.

Even when a process and related artifacts are well understood sometimes people choose not to follow it, either individually or as a group. Most processes have some room for skipping steps and leaving out artifacts for more simple scenarios. The danger is skipping steps that are not well understood, and not compensating for the side-effects that would be expected by skipping that step. In cases of larger projects and more people this is almost never a good idea and the side-effects result in escalation to large problems that lead to an ineffective design and implementation that doesn't meet the expectations of the expert users, possibly even because of a failure in the beginning to capture those expectations.

An interesting combination of the two sides of this comes up when there is limited understanding and lack of agreement because of misconceptions about a process and/or artifacts. Without understanding the big picture and how to go about working within it the natural tendency for a person is to fall back to whatever they are most familiar, or to perform according to whatever makes the most sense to them. This can even take the form of resistance to the process and artifacts, and based on that some ongoing undermining of general efforts as well as not producing artifacts (deliverable) consistent with others on the team.

When All You Have is a Diagram Everything Looks Like a Nail

Strengths and Weaknesses of Diagrams

Diagrams are wonderful tools for communicating certain types of concepts, especially when focusing on one aspect or another of a larger concern. They are inherently low on detail and good for enabling understanding of relationships between diagrams elements, and in some cases even characterizing those relationships.

The definitions of the elements and relationships between them generally need to be defined elsewhere to keep the diagrams concise. These are often necessary for the understanding of a diagram.

The downside of diagrams is that it is very easy to create a diagram that implies things that are incorrect, and most diagrams are impossible to understand correctly without some narration to describe how it is organized, and as mentioned above the meanings of the different elements and relationships.

There are various diagrams described in this process that are good uses of diagrams, with detail for diagram elements in separate documents. These include the Use Case Diagram and the Screen Flow Diagram.

For many of the other artifacts it may seem natural to try creating a diagram instead of writing text, and even for those artifacts where a document is a natural fit it can be tempting to create only the diagram and not the supporting document. In those cases if the diagram is done well a high level structure might be documented in it, but the supporting details required to understand it, and required to be able to act on it from a systems development perspective are nearly completely missing. This results in diagrams that are somewhere between not useful and harmful because they imply things that are incorrect without supporting details and it is easy for people to misunderstand them, often without realizing they have misunderstood.

Diagrams for Requirements Gathering

A common area where there is a natural tendency to create a diagram instead of writing text is in the requirements gathering efforts where stories and statements should be used. In addition to the general problems with diagrams and a lack of supporting documents there are additional problems with using diagrams for requirements gathering:

- there are many details that need to be documented when gathering requirements and diagrams are too simple to capture them
- diagrams do not transition well into Use Cases (whereas process related stories do)

because the level of detail in diagrams is too high when one person uses them to feedback what they heard to the person specifying requirements there is a lot of room for misunderstanding