Hope, Belief and Wizardry

Three different perspectives on project management

Markus Völter
voelter@acm.org
Version 3.0, June 18, 2004

Introduction

Normally, I'm not too much into project management. However, over the last couple of years, I have come across several projects that use hope, belief and wizardry as their primary management tools. You will probably not have heard about these specific projects – and you probably never will. They have gone out of existence, maybe because of too successful management techniques. However, if you're a consultant or a developer, I'm sure you have experienced your own hope/belief/wizardry project and you could probably tell the same story as I do here.

This paper is mostly a product of personal frustration. I simply had to write down these things. I did not include "three known uses", as you might guess, because the cited projects and their staff might feel offended. However, for me there is no doubt that these things here are really patterns. I have seen many instances, and you have probably also come across some. The pattern form used in the paper consists of a narrative, a kind of story, interleaved with short pattern thumbnails that capture the essence.

So, why the title *Hope, Belief and Wizardry*? Quite simple, these are the three perspectives how the different parties see the project:

- The customer often does not really understand how management and the developers try to make the project a success. The customer is typically full of *hope* that the project will eventually succeed.
- Project management is usually very convinced of what they do. They have a firm *belief* that they will successfully steer the ship through the stormy seas, finishing on time and in budget.
- And the developers would consider it *wizardry*, if all that really worked out in the end.

Of course I know that nobody would actually set out to run a project in the described way. However, strangely enough, I've been on several such projects as a technical consultant. And yes, we all know how to run project more efficiently, there are many methodologies, processes and practices described

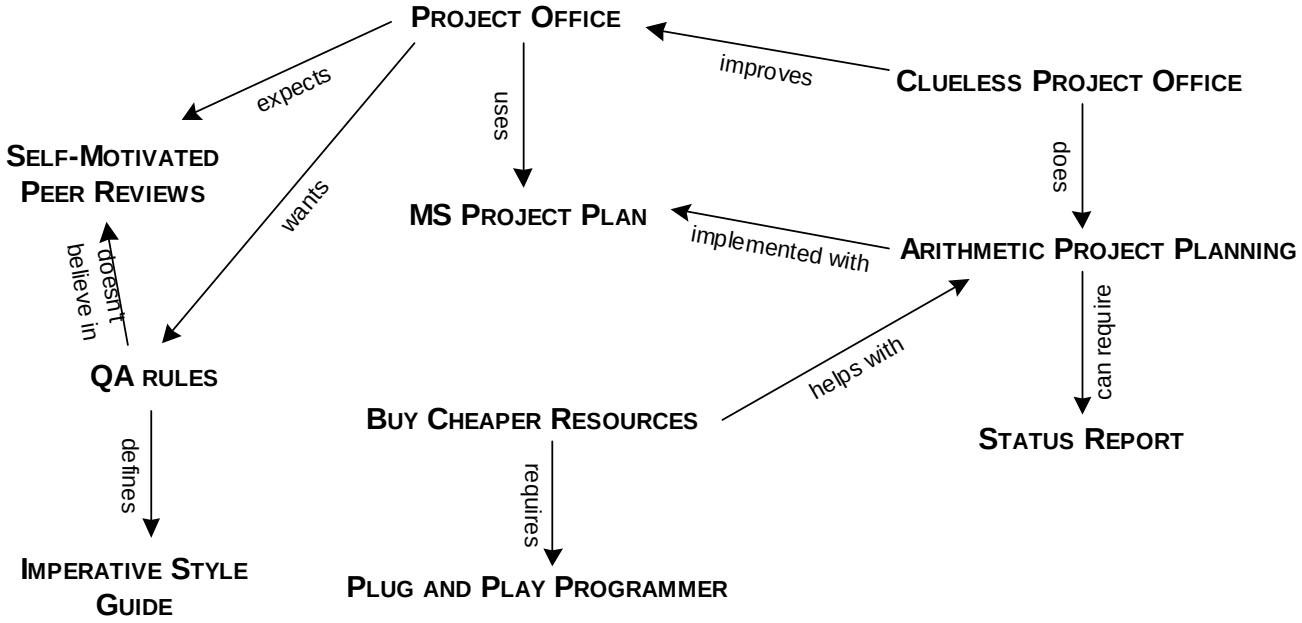
that help here and try a more people-oriented approach to project management. But still: Many projects today are still managed "the old way" and maybe this paper can help to show how absurd all this is, maybe by sometimes exaggerating a little bit ☺

Disclaimer: Of course, you should not really take these patterns as *good* advice, as they are cynic and ironic. It is up to you, the reader, to spot the irony and judge how successful the patterns can be used in muggle software development projects. And of course, the observations described in the patterns are not new, or even described for the first time. Many are closely related to agile processes: using agile processes is a good way to omit stumbling into a hope/belief/wizardry situation.

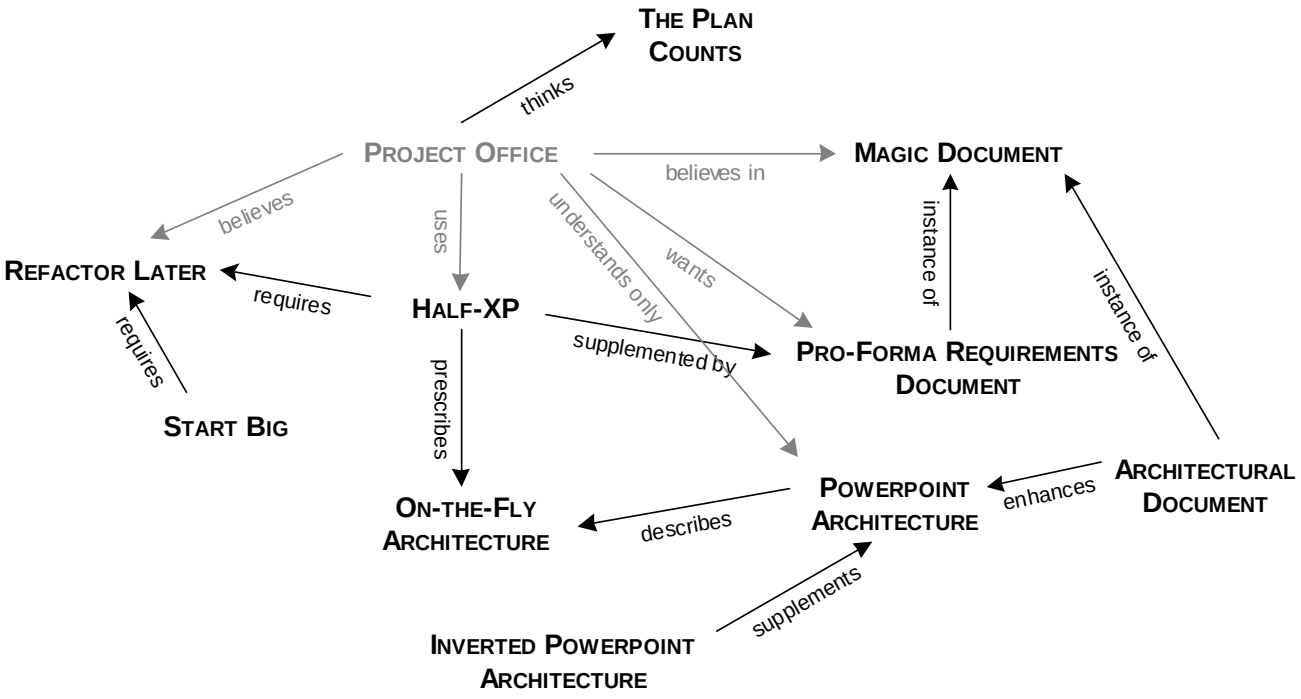
Overview over the patterns

The overview is presented in the form of diagrams showing the patterns and their relationships. The patterns are organized into several subsections, each of them has its own diagram. Patterns introduced in a previous subsection are rendered in gray.

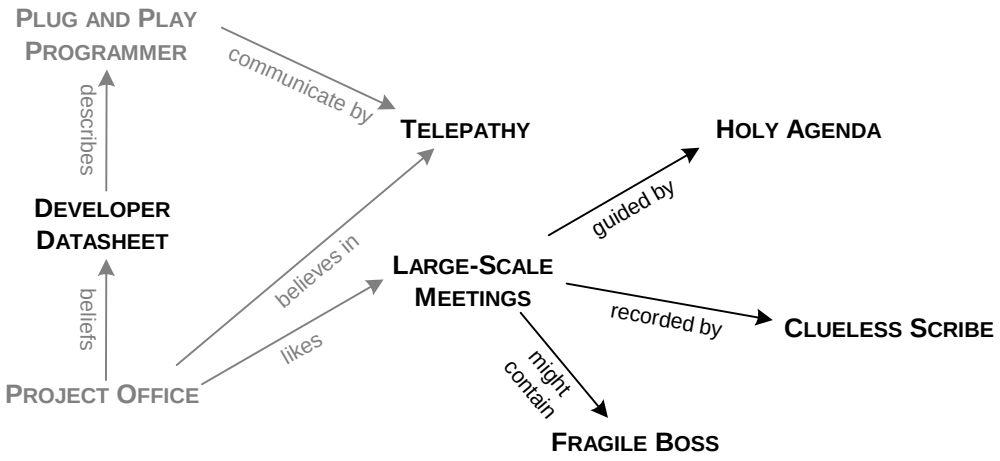
When starting a project, you will start with **project planning.** During the course of the project, **controlling** becomes a more and more important aspect. They following patterns apply to this domain:



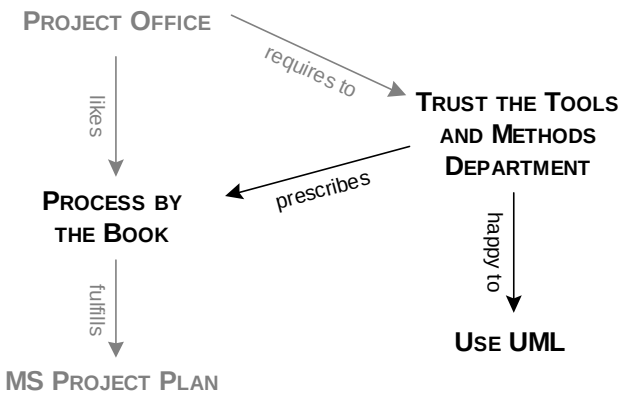
During the project, you will have to deal with **requirements, architecture and design** aspects. We also have some patterns there:



The **development team** also deserves proper attention and management – we offer some help in these patterns:



Last but not least, **methods and tools** are an important aspect of each project and must be given their necessary attention.



Setting the stage: The projects

The projects I'm about to tell you have happened in completely different domains. Some where rather typical e-commerce projects, others are from the financial domain, and some have happened in industrial environments. While this is rather diverse, all the projects have a set of common characteristics: they were rather large software development projects ranging from 10 to 150 developers plus a (more or less) corresponding number of managers and administration people. And in all the cases, in addition to just fulfilling the requirements of the project at hand, the architecture developed as part of the project should be reused in future projects, it was going to be kind of standard architecture for the respective enterprise.

The Patterns

Patterns for project planning and controlling

> The project is important. Because of this, we have a couple of administration people assigned to support the project. They are not really part of the development team, they are located in a so-called project office. These people are very experienced. You can tell that from the fact that they are rather old, and that they come from big companies with big, well-known names (or, they are really young, qualified with MBAs, eager for promotion within their organisation, and therefore they must be good). Their job is to control the project and provide management and planning support. Because they are not part of the development team per se, they can successfully work on their own and look at the project from a different angle. And they are not annoyed by day-to-day development work! This allows them to work very efficiently.

PROJECT OFFICE

Context: You are running a rather big development project.

Problem: How do you make sure the project is controlled an planned properly?

Solution: Provide a project office staffed by people who are not part of the development team. This allows them to see the project from their own point of view. Management can be assured of their experience and competence by ensuring that they come from a big-name company. The pattern can be implemented most efficiently by actually using a CLUELESS PROJECT OFFICE.

The primary job of the project office is to control the development team. This job is simplified because they are not part of the team and consider themselves somewhat superior… anyway. You might argue that such a separate, remote project office is no good, because they cannot work effectively with the project team. This is far from true! Being on the team would not help them, because these people should not have a clue about software development or the requirements the software should implement. Such knowledge would even limit their ability to view the project from a different, rather formal angle. They should not be distracted by the day-to-day technical details of how this project is run or the business it should support. They should just control, on an abstract level, that everything goes as planned.

CLUELESS PROJECT OFFICE

Context: You have a PROJECT OFFICE in place that controls your project and makes sure it stays on track.

Problem: How do you ensure they can work efficiently and are not too much bothered by the details and day-to-day problems of the project team?

Solution: Make sure the people in the PROJECT OFFICE don't know anything about software development, or even the requirements the project is about to fulfill. Make sure the PROJECT OFFICE does their work on an abstract level. This allows them to work more efficiently, and it ensures the "different angle" mentioned in the PROJECT OFFICE pattern.

So, the question arises, what these people actually do? What are their tools and how do they communicate with the development team? In most projects this is rather hard to determine, but you will see that there are well-known and proven solutions to all these problems.

Their main tool is the MS PROJECT PLAN. This plan provides a concise, comprehensive and complete overview about all aspects of the project. At least about all those aspects that are ultimately important. It intentionally ignores these low-down details of those who work in the development team. It is the ideal abstract tool for people who cannot care about the details, because they have to control the big picture and the overall success of the endeavour.

MS PROJECT PLAN

Context: You already have a CLUELESS PROJECT OFFICE in place.

Problem: The PROJECT OFFICE has to keep track of all the important activities, people, resources, milestones, etc. This has to be fairly abstract, though, because they cannot afford to be distracted by what they perceive to be inappropriate concrete detail

Solution: Use a colorful MS PROJECT PLAN to provide the overview. This vehicle is ideal for that purpose because it is easy to use for the PROJECT OFFICE staff, it provides a nice, management-compatible way to graph things, and it is abstract enough to ignore all the gory details of everyday life in the project.

You might say that important details might get lost in such a high-level view, or worse, that some of the requirements might not be fulfilled or that the development team will not be able to work with these MS PROJECT PLANS. But note here that the development team, of course, will never see these plans anyway. Developers are only hunted down by the PROJECT OFFICE, being told that they are behind the plan – but they will never actually see it. So how, you might ask, do those people in the PROJECT OFFICE get the information they need to build this MS PROJECT PLAN? One possibility that's regularly used is that they just make it up by themselves. To make sure that the MS PROJECT PLAN contains at least information that it logical in itself, they can do some simple arithmetics, based on the available amount of time till project deadline, the available resources and the available money.

ARITHMETIC PROJECT PLANNING

Context: You already have a CLUELESS PROJECT OFFICE in place using a MS PROJECT PLAN as their primary planning tool.

Problem: You, the CLUELESS PROJECT OFFICE, have to fill content into the MS PROJECT PLAN. You want to do so without much interference with the development team. You need a way how the plan can be populated by only talking to management (or to nobody at all). And the plan must look consistent as a result.

Solution: Use simple arithmetics to fill in the MS PROJECT PLAN. You know when the project should be finished, and you know, how much money is available for it. Thus, in a first step, calculate how many resources you can afford:

numberOfResources = availableMoney / averagePricePerResource

In the next step, you can then determine who does what, and when. Based on the requirements and the available time till the deadline, you can easily draw a nice-looking MS PROJECT PLAN. If the result looks unrealistic, BUY CHEAPER RESOURCES. If the PROJECT OFFICE staff needs information from the developers they should avoid direct contact and instead use STATUS REPORTS.

This approach works well, because the PROJECT OFFICE don't need to get in touch with the people from the development team. And, in

fact, the plan looks really good, because it meets the deadline precisely and does that even with the allotted money! Perfect.

In the course of the project, the MS PROJECT PLAN needs to be updated! Fortunately enough, the approach described in ARITHMETIC PROJECT PLANNING also works repeatedly during the project. An experienced PROJECT OFFICE team can update the plan on their own, still meeting deadlines and using only the available money. A PROJECT OFFICE with less experience has a problem, though. They need to know how much work has already been done. Getting this information is not possible without some contact with the development team. To avoid direct contact, however, and to allow the CLUELESS PROJECT OFFICE staff to understand what the development team tells them, use well-organized forms, or reports, which the development team members have to fill in regularly. They require the development team to abstract to a level that's understandable by the PROJECT OFFICE.

STATUS REPORT

Context: The PROJECT OFFICE needs to update their MS PROJECT PLAN during the project using ARITHMETIC PROJECT PLANNING and does not have the experience to do this without any contact to the developers.

Problem: To be able to update the plan, inexperienced PROJECT OFFICE staff need information on the progress of the project from the development staff. This information needs to be stated in a way that is understandable for the CLUELESS PROJECT OFFICE.

Solution: Make each member of the development team fill in a status report form every once in a while, for example every week, on Wednesday at 4pm. In this report, he is required to state the progress he made, report problems, and describe what he'll do next. If a member fails to finish with what he planned, require him to explain why. Ideally, add a "blame field" to the form where he can write down the name of the colleague whose fault it is.

Using these STATUS REPORT forms, the PROJECT OFFICE can easily track the progress of the project. In combination with ARITHMETIC PROJECT PLANNING, this can result in an always-up-to-date MS PROJECT PLAN. However, towards the end of a project, the PROJECT OFFICE team might find out that more and more remaining work has to be done in an increasingly shorter time frame until the fixed deadline. They might be tempted to talk to the customer to find a solution, such as reducing the scope of the project, reprioritizing requirements use cases or even spending more money. However, as it turned out over the last 20 years of software project management, this usually doesn't work. But, there is another solution: just replace your resources by cheaper ones. Then you can buy even more, and

the project will proceed more quickly. Take a look at ARITHMETIC PROJECT PLANNING if you don't believe this.

BUY CHEAPER RESOURCES

Context: You get to the end of the project and ARITHMETIC PROJECT PLANNING together with STATUS REPORTS reveals that you probably won't finish in time…

Problem: How do you still finish the project in time without reducing the scope or getting more money and without having to discuss with management?

Solution: The solution is to use more, but cheaper resources. ARITHMETIC PROJECT PLANNING reveals this as an effective way to increase development speed. The only thing that might eventually suffer is quality – but that's not something that shows up in your MS PROJECT PLAN anyway.

BUYING CHEAPER RESOURCES to replace expensive ones is a rather effective way to stay within the MS PROJECT PLAN and the budget. Especially, towards the end of the project, this can be a real "life saver". Introducing new, cheaper resources into the project towards the end is without risk. Good people can start work and be efficient right from the start, so-called PLUG-AND-PLAY PROGRAMMERS.

PLUG-AND-PLAY PROGRAMMER

Context: You BUY CHEAPER RESOURCES to help you in the tight schedule towards the end of the project.

Problem: How do you actually make sure that the new, cheap resource works efficiently right from the start?

Solution: Make sure that you only buy cheap resources, you also need to make sure that they are actually so-called plug-and-play programmers. Those are characterized by the fact that they start to be productive in any project right from the start. They don't need time to familiarize themselves with code, tools and the project. Also, you don't need to coach them!

Using PLUG-AND-PLAY PROGRAMMERS has the additional benefit that they don't need to be coached. In the usually tight timeframe towards the end of the project you don't have resources to bring new ones up to speed – after all, resource shortage is the reason to actually BUY CHEAPER RESOURCES. If you're not sure whether a potential new resource actually is a PLUG-AND PLAY PROGRAMMER, take a look at their respective DEVELOPER DATA SHEET.

Now, as mentioned, quality might become an issue. But how can you actually measure something as diffuse as quality? One possibility to avoid the need for such measurements is to make sure from the outset that code quality is rather good. You can easily ensure this by using peer reviews. Peer reviews are conducted by the development team, internally, and are used to ensure code quality. Because

developers take pride in what they do, they will execute these reviews automatically, all the time, even under rigid time constraints imposed by the PROJECT OFFICE and their ARITHMETIC PROJECT PLANNING.

SELF-MOTIVATED PEER REVIEWS

Context: You run your project using ARITHMETIC PROJECT PLANNING.

Problem: How do you ensure the quality of the code?

Solution: The code quality is ensured automatically by the developers. Because they take pride in their work, they will conduct peer reviews with each other. This happens even under the severe time constraints that result from ARITHMETIC PROJECT PLANNING, and it works even with the recently BOUGHT CHEAPER RESOURCES.

As mentioned, these reviews go without straining the projects time and resource budget because they are just done as part of the everyday work of the developers. No consequences for the MS PROJECT PLAN! All this self-motivated stuff on the developers side might seem suspicious to the PROJECT OFFICE, though. They don't believe in motivated people... doing the right thing without pressure and control. A controlling agency, called Quality Assurance, is therefore necessary to ensure the quality of the developed artifacts. The QA people are usually associated with the PROJECT OFFICE but consists of (former) developers – they thus have a thorough understanding of the project's requirements, the used tools and they are experienced in reviews and the like. Therefore, QA is always the last, finally deciding instance in a project.

QA RULES

Context: Your PROJECT OFFICE does not completely trust development team.

Problem: How do you ensure the quality in the face of mistrust? And how do you ensure homogeneous quality and the obedience to standards all over the project?

Solution: Put a quality assurance team in place. It's task is to ensure quality of the development artifacts on an abstract level. They usually run reviews, interviews, etc. and they are typically associated with the PROJECT OFFICE. They have deep insight into the project and its constraints.

To to make reviews effective (by the QA or by peers) the developers have to be able to understand each other's code (this is even true for PLUG-AND-PLAY PROGRAMMERS). Developers are usually a funny bunch of people. Everybody uses their own weird style of arranging the code. Specific problems arise in C-like languages with the position of the opening brace (end of previous line, or beginning of next line). Naming of variables is also a problem. Purists claim (correctly) that variables can be as short as they like, as long as

distinct variables have unambiguous naming (for the compiler, that is!). Others want variable names that have 70 characters in order to convey their complete semantic meaning. But, good luck we have QA people. For them, all this is no real problem. They just devise a style guide for the developers to follow – which they do happily because they are told to do so.

IMPERATIVE STYLE GUIDE

Context: You want to ensure code quality by using SELF-MOTIVATED PEER REVIEWS.

Problem: How do you make sure everybody can understand everybody else's code?

Solution: Let the QA team define a style guide. The guide contains everything from code layout to variable names to documentation requirements. Place this MAGIC DOCUMENT somewhere into the intranet, the developers are happy to read it and follow it promptly. Make sure that developers cannot easily change or adapt the document, because QA RULES.

Patterns on requirements, architecture and design

Before you actually start coding in a project, you have to have some preconditions met. That's something everybody knows, of course. For example, you need to fix your requirements. A document needs to be signed by the customer that specifies every single bit of functionality that has to be implemented in the project. However, people found out that this is unrealistic, for several reasons: First, the customers usually do not know, what they want. Consequently, they are unable to write (or at least, sign) a requirements document. Second, the requirements might change over the course of the project, so a requirements document would change all the time anyway. And third, if a document that contains all requirements would be used as a basis for a vendor's offer, the price would be too high for purchasing to accept…

That's why people proposed to use a more agile process, where you have fixed time, resources and quality but where you are free to adjust the scope – of course only after talking back to the customer. This is no problem, because the customer is always available on the project and can answer questions about what to implement first, and what to postpone. So the best approach is obviously to combine the two approaches to have the best of both worlds: no fixed requirements, but you still want to make sure that the to-be-done software does everything the customer requires (and a bit more) with a fixed budget at a fixed deadline.

HALF-XP

Context: You are not able to clearly define the requirements of the project from the beginning.

Problem: How do you still make sure that the customer gets all he wants with a fixed budget and a rigid deadline?

Solution: Use half of the XP methodology. Do not define the detailed requirements up front, start developing immediately. The developers will refine the requirements as they go by talking to the customer's representative. It's not necessary, though, that the customer representative is available *all the time* on the project, because the customer promised to be accessible whenever he's needed. Because you have a rough understanding on what the customer wants, it's easy to finish with the project on time and meet the customers requirements.

So, as we can see, it does not really make sense from a technical perspective to fix the requirements upfront. HALF-XP still allows you to run the project efficiently. Problem is, that there are those other folks at the customer's – those folks in the purchasing department. They don't know about fancy XP or agile methodologies. They want to buy software the same way as they buy, say, twenty-five-thousand screws. So they want a clearly defined lot, and a fixed price (fixed meaning: fixed after they have bargained the initial price by at least 15%, no matter what the initial price was – that's how purchasing people earn their bonuses). So, formally, you have to sign a requirements document anyway. You'll probably never look at it again after it is signed, but you have to sign one. And its content is probably not really useful…

PRO-FORMA REQUIREMENTS DOCUMENT

Context: You use HALF-XP but the customer's purchasing department still wants a formal requirements document.

Problem: How do you state formal requirements if you don't know them?

Solution: Write a pro-forma requirements document. It includes everything, but described in very general and weak terms. This document can be signed easily. Nobody will ever look at it again, and everybody knows, that the real requirements will be defined on the fly using HALF-XP. The purchasing department is happy, though.

So, everybody is happy: The purchasing department is happy, because they have requirements and can purchase software using the same approach as for screws or sacks of cement. The "real" customer is happy, because a vendor has been found (with a happy purchasing department), knowing that the real requirements will be worked out on the fly (and they will contain everything they ever wanted). And the vendor is also happy: He has got a contract and the real requirements will be worked out on the fly (and they will contain

only the bare minimum that's absolutely necessary). The legal departments will fight out differences.

Now comes the PROJECT OFFICE, however. They don't know of anything but the PRO-FORMA REQUIREMENTS DOCUMENT. And they need to create their MS PROJECT PLAN. They will be happy to use the PRO-FORMA REQUIREMENTS DOCUMENT as a basis for their ARITHMETIC PROJECT PLANNING…

Now, after long last, you actually start coding. The customer is looking forward to the first live results of the project. He wants to see something for his money: code. Ideally, running code. And even better: *useful* running code! So you have to start working to convince the customer he has chosen the right vendor. To get something done quickly, you should work with as many people as possible.

START BIG

Context: You have just won the contract and want to provide useful code to the customer as quickly as possible.

Problem: How do you get something done as quickly as possible?

Solution: Start big. BUY as many CHEAP RESOURCES as you can get right from the beginning. The beginning is where the hard work has to be done: frameworks, base libraries and other "strategic" code. You cannot have enough people to work on that critical phase in the project.

So you have a wealth of resources working on your most important problems. Quickly you will have something to show your customer. Looks nice. However, to prevent the project from drifting into chaos, you need an architecture and some structure. As an experienced project manager, you know that, of course, and that's why you set up an architecture team. This team consists of your most skilled resources. However, architecture is not something the customer can see. And it's not part of the MS PROJECT PLAN, anyway. So, to convince the customer of you, you have to implement "user features" and create an architecture on the fly.

ON-THE-FLY ARCHITECTURE

Context: You have STARTED BIG and you need to provide an architecture for the system.

Problem: How do you define and implement an architecture while the project is in its initial phase and the MS PROJECT PLAN does not explicitly allow time for the architecture?

Solution: Implement the architecture and the basic libraries in parallel with the first user features. Your best resources will define the architecture and talk to the rest of the team about what they should implement, and how. It is not necessary to define the architecture (or at least, an outline) up front because you can REFACTOR LATER. Note

that this pattern works especially well for mission- or safety-critical enterprise projects.

However, sometimes your customer has so-called technical managers – people who have been developers or techies in their earlier lives, and who have advanced their career into management. Usually, they don't have any clue about current software technologies, but they think they are experienced ex-techies and want to be consulted for technical decisions. Thus, you have to make sure they like the system architecture, or what they think the architecture should be. Therefore, provide simple a overview chart, with at most 7 boxes[1] and some connecting lines. Make sure these boxes contains terms they understand and that enough technology buzzwords are mentioned. It is crucial that the lines have no defined semantics because this will limit your freedom in implementing the actual system.[2]

POWERPOINT ARCHITECTURE

Context: Your customer has some kind of technical managers.

Problem: The customer requires to present them with the architecture. The audience has some technical background but is by no means up-to-date or competent with current technologies – however they usually know management-compatible buzzwords.

Solution: Create a small Powerpoint presentation that shows the system as a collection of at most seven boxes connected by (ideally unannotated) lines. Make sure the presentation is colorful, contains some well-known terms from the business and mentions all the current buzzwords.

Using this pattern is not without risk, though: There are those technical managers who think (correctly) that you cannot represent a system architecture using seven boxes and a couple of connecting, unannotated lines. They want to see the full complexity including every nitty gritty detail. Not that they understand it – but they want to be exposed to "the real thing" and impress others with complicated diagrams. Once you found out about this, you can easily switch to using the INVERTED POWERPOINT ARCHITECTURE pattern:

[1] It is generally accepted that people can remember up to 7 items easily.

[2] As part of UML 2.0, several groups of people are working on enhancing the semantics of UML and its metamodel to allow direct execution of UML models – some people call this executable UML. We propose another direction for improvement called *Executive UML*: The notation is reduced to only boxes and lines with *no defined semantics* at all to be suitable for direct understanding by executives.

INVERTED POWERPOINT ARCHITECTURE

Context: Your customer has some kind of technical managers.

Problem: The customer requires you to present them with the architecture. The audience has some technical background but is by no means up-to-date or competent with current technologies. However, they want to see the full-blown details of the upcoming system, usually to impress colleagues with complex diagrams.

Solution: Create a *large* Powerpoint presentation that shows the system in as much detail as possible, using diagrams with collections of *at least* seven boxes connected by lines, annotated with well-sounding terms. Make sure the presentation is complicated, lacks color, and contains important-looking abbreviations, some well-known terms from the business and all the current technology buzzwords. And USE UML!

It is well possible that the POWERPOINT ARCHITECTURE is the only architecture specification you will ever create because no resources are there to define a real system architecture. If you're lucky, however, you will eventually get the time to *really* document your architecture. In some projects, such a document is even a deliverable, which means that it *has to* be created and delivered[3] to make the project a formal success. This is good for the architect. He will have time (based on the MS PROJECT PLAN) to design and document an architecture. Once this is done, however, the document is usually stored somewhere and never touched again. But you still expect that every developer reads it, and follows the architecture automatically.

ARCHITECTURAL DOCUMENT

Context: You have to provide the system architecture as an explicit deliverable.

Problem: How do you make sure the architecture is really implemented and followed allthrough the development process?

Solution: Write down the architecture in a ARCHITECTURAL DOCUMENT. Publish this MAGIC DOCUMENT to every developer in the project. They will happily stick to it and implement the architecture consistently. You don't need to provide examples, execute architectural reviews, or explain the architecture to the developers individually.

Because developers always discover, read and follow MAGIC DOCUMENTS, the system will be clean and well structured, just as described in the ARCHITECTURAL DOCUMENT. In case the MS PROJECT PLAN does not give you the time to write an ARCHITECTURAL DOCUMENT, using ON-THE-FLY ARCHITECTURE together with START BIG might result in unstructured, even chaotic code in the beginning. QA will not like that, and the developers won't like that either. During their SELF-MOTIVATED PEER REVIEWS they will find out about

[3] At the beginning, of course, and then it must never be changed again!!

this chaos. They will want to fix, refactor these things. But to do that they need time. But there's no problem, because developers will get that time later, certainly (if there is not time, BUY more and CHEAPER RESOURCES).

REFACTOR LATER

Context: Your ON-THE-FLY ARCHITECTURE is late and you have to keep implementing customer features.

Problem: How do you make sure the code does not drift into complete chaos, and the architecture is really implemented?

Solution: Defer refactoring till later in the project, when the customer is convinced that you are a good team. The customer will give you time and resources later, because the customer is interested in good product (code) quality. If you implement many features in the beginning, the MS PROJECT PLAN will reveal enough free time for refactoring towards the end of the project.

Project management is all about balancing different interests. The different parties involved define *success* is different ways. Developers, for example, usually define success as "having had the chance to play around with a lot of fancy new technologies". The vendor, usually, is happy when they earn a lot of money while delivering as little as possible. For the customer, it's the opposite way: little money for a lot of delivery. To resolve this, the only unbiased judge is the PROJECT OFFICE and it's MS PROJECT PLAN.

THE PLAN COUNTS

Context: The project is nearing it's end and you want to determine whether the project is a success or not.

Problem: How do you define success? Every party in the project has a different definition what makes a project successful for them. You have to find an unambiguous way to define success.

Solution: Success is, when the MS PROJECT PLAN is fulfilled. When the CLUELESS PROJECT OFFICE created the MS PROJECT PLAN, it took into account *all* important issues and requirements. Obviously, when the plan is fulfilled, the project is a success.

As a sidenote, if you look more closely at the MS PROJECT PLAN, IMPERATIVE STYLE GUIDE, PRO-FORMA REQUIREMENTS DOCUMENT, ON-THE-FLY ARCHITECTURE and MS PROJECT PLAN you can see a common (meta-)pattern emerge from all those patterns which is called MAGIC DOCUMENT:

MAGIC DOCUMENT

Context: You have something that everybody needs to know or adhere to.

Problem: How do you make sure that these things are known to everybody, and are actually followed?

Solution: Write these things into a document. Pass this document to everybody. The MAGIC DOCUMENT will make sure by itself that it is implemented, discussed, followed, or whatever else should happen with the content. Once it's written and published, you don't need care about it anymore.

You can even generalize this one step further, resulting in the *Publish and Forget* philosophy, which says that all you have to do is publish something, and then forget about it. The rest – discussions, implementation, etc. – will happen automatically.

The development team

Unfortunately, software development has not yet been completely industrialized. To create software, you need people. To hide that, and to make managing these people somewhat less of a "personal" thing, these people are usually called *resources* in a project, just as rooms, computers, money and the like. A resource is characterized by the set of skills that it provides. For examples, a computer has 256 megs of RAM and a 20 gigabyte hard disk. A developer resource can program Java, or draw fancy little pictures with Rose. And just as there is a datasheet for a computer that specifies its performance data, there is a datasheet for developer resources. It is called a CV and lists all the skills of the resource. So in order to acquire the resources for filling-in the MS PROJECT PLAN, the only thing you need to do is select appropriate resources based on their datasheet.

DEVELOPER DATASHEET (AKA CV, PROFILE)

Context: You need to acquire developers for your project, primarily to fill in the MS PROJECT PLAN.

Problem: How do you make sure you find "compatible" resources providing exactly those skills you actually need for the project?

Solution: Base your selection on the developer datasheet, also known as CV or profile. This document describes all the skills the developer has, as well as his experiences. The content of these CVs is always true and can be trusted, because they are usually not specified by the developer himself, instead these things are stated by his employer. The CV does not contain any personal or social skills, but these are not necessary for developer resources anyway. Their technical skill is what counts.

Using the DEVELOPER DATASHEET pattern will save you a lot of time because you don't need to interview all those potential resources personally. After all, it's not a project manager's job to fiddle around with people – as the job title *project manager* implies, he has to deal with the *project*! Note that good DEVELOPER DATASHEETS also contain

information about the social skills of a developer, allowing you to judge if he fits the team. Such social information is usually given by terms such as *good team worker, communicative* or by a list of social-skills-trainings the person has had.

Once the project has started, the developers have to talk to each other to make the project become a success. They have to exchange their thoughts on the system, on problems, and so on. This is especially important in larger projects, where it's impossible that everybody sits in one room. Good luck that the crew of *USS Enterprise* have found out about a specifically efficient means to communicate: telepathy. Information is directly transferred from brain to brain, without the semantics-filtering detour of spoken language. And it has even more advantages. It works from one room to another, and even if the project is distributed over several buildings, locations or continents, telepathy works.

TELEPATHY

Context: You have your resources available and the project is running.

Problem: How do you enable efficient communication among developers in the face of distributed development sites?

Solution: Use telepathy as the basis for the communication among the developers. It works over long geographic distances and transports thoughts directly without having them to wrap with semantics-filtering language. In projects where there are several languages spoken by the developers, this is especially an advantage, because translation is not required.

Using TELEPATHY also has the advantage that you don't need an expensive and complex support infrastructure, such as computers or telephones. It works by building a spontaneous peer-to-peer system using the developers as network nodes. However, there are people in the project who cannot effectively communicate that way: typical examples are management, the PROJECT OFFICE, and also some less experienced developers. They insist on using the more traditional means of communication, such as …. Meetings!

Because people sit in close proximity to each other, they don't need telepathy to communicate, they can use the spoken word. Usually, these meetings are arranged by the CLUELESS PROJECT OFFICE. Because they typically don't know who is responsible for what, they will usually invite more or less everybody to join the meeting. But because meetings are rare events it's a good idea to make sure that all relevant people are there.

LARGE-SCALE MEETINGS

Context: You want to share information to many people – discussion seems necessary.

Problem: How do you make sure that in meetings, all the relevant people are actually really there?

Solution: Invite everyone who seems even remotely concerned with what is discussed in the meeting. People, especially developers, like meetings because it's there where the biggest progress is usually made. Those who have nothing to contribute can still serve as a consumer of the cookies and the coffee that's usually served.

Successful projects can be distinguished by the number of effective meetings they use. Another motivation for LARGE-SCALE MEETINGS is that they are a power trip for the PROJECT OFFICE: it shows the team how big they are, and how many resources are under their control. It is also likely to impress the customer if they know LARGE-SCALE MEETINGS take place. Since meetings are so important and commonplace in business, I want to elaborate some more on them.

Most meetings have an agenda. At least, it is a good idea to have one, because otherwise everybody will just spend their time drinking coffee and eating butter-brezels until the scheduled meeting time is over. An agenda can help to make sure useful things are actually happening in a meeting. An agenda is even so important that you should stick to it strictly. Even when you discover more important things, or the problems on the agenda vanish for some reason, you should still stick to the agenda. After all that is what's planned, and that is what people expect!

HOLY AGENDA

Context: You prepare an agenda for a meeting to make sure everybody knows what to expect and how to prepare (although nobody will actually prepare!)

Problem: What do you do if the content, priorities or the problems change even during a meeting or directly before?

Solution: Stick to the agenda. An agenda is a kind of "law" for the meeting. It must not be changed. You should even stick to the agenda if it has become completely outdated.

There is more to say about meetings. For example, typically a meeting needs to have meeting minutes. Reason is that even people who could not attend the meeting want to know what happened. Remember that meetings are important, and everybody want to know who has been at a meeting, that's typically the only thing people read in meeting minutes. However, writing the minutes is something rather annoying. Nobody really wants to do it expect … except for the people from the CLUELESS PROJECT OFFICE. They are happy to do that because they have nothing else to do anyway. And

it adds to their heap of paper (whose size is another heuristics for the importance of the project). However, PROJECT OFFICE people typically have no clue about the contents, which either results in meaningless minutes or repeated questions for clarification from the scribe. This is good because it helps to describe things clearly and unambiguously.

CLUELESS SCRIBE

Context: You are running a meeting, and minutes are required.

Problem: Who do you select to write the minutes?

Solution: Use a member of the CLUELESS PROJECT OFFICE. They will have to ask questions for clarification of issues all the time. Although you might think these are annoying, in reality they help to clarify things and come up with clear, useful and precisely worded minutes.

There is also an important social aspect to meetings. In many cases, a meeting is just run for some kind of boss. They are some kind of play that reiterates a discussion that has been run before among techies, for some kind of decision maker to attend and "understand". Now, usually, the decision maker is important. And thus you have to treat him (or occasionally, her) with care. One thing they are typically allergic to is truth: truth about schedule, the project plan, technical problem, or the truth about their making decisions for political reasons… Meetings can end rather quickly if they are confronted with the truth. Therefore: avoid that. Tell decision makers just what they are prepared to hear, and what you know they can handle.

FRAGILE BOSS

Context: A decision maker is sitting in a meeting "to learn how things are going".

Problem: What do you do if things are not going well, or if you need to discuss/say things that might not be in line with what the boss expects (or has been told before the meeting by the CLUELESS PROJECT OFFICE?

Solution: Lie. Tell him what he wants to hear. Never confront him with stuff of which you think he does not like. Treat bosses with care and make sure they feel comfortable in meetings.

Methods and Tools

While you have to deal with people in a project, the more important aspects are tools and methodologies. People are merely needed to operate the tools and play the roles defined in the methodologies – to date, no way has been discovered to run projects without people. However, watch out for the tools you use! Good luck, in most larger organizations, there is a dedicated department that that deals with

tools and methodologies. They are staffed with people that have lived through (and survived) many real-life projects and are happy to let you benefit from their practical, real-world experience. Usually, they do this rather indirectly, by prescribing technologies, tools, processes, document templates and clothing style using MAGIC DOCUMENTS.

TRUST THE TOOLS & METHODS DEPARTMENT

Context: You want to run your project efficiently and in line with company standards.

Problem: How do you make sure you use the best tools available, the best technologies, and the most efficient processes that have been proved throughout many projects?

Solution: Trust the Tools & Methods department. They are staffed with highly practical, experienced and skilled people and are happy to help you pragmatically with your project's considerations.

It is not appropriate to have a toolsmith on the project. The reason is that this toolsmith might be tempted to select tools and methods that are focused on the specific needs of one project instead of focusing on company standards and broader strategic or business considerations

Standards! Standards! When running a project, you should make sure that you use as many standards as possible (except, perhaps, in the situations where two standards actually contradict each other – if you find out about it!). This is not just true for the tools and methodologies mentioned above. It's also true for reference architectures, languages, processes, etc. It's usually a bad idea to adapt standards to your specific project's needs – the full power of standards will not come to you! Adherence to standards is more important than pragmatic decisions helping the project. Ask the QA people, they will confirm that! There are many examples for successful use of standards, I will focus on one very popular one, USE UML[4].

USE UML

Context: You want to represent something graphically.

Problem: You don't know which notation to use for your graphical representation, but you know that you should adhere to standards.

Solution: Use UML. Thank god, UML is extensible and can therefore be adapted to represent whatever concept you require. Because you use a standard, it's easy for readers to understand what you want to convey.

[4] Related ones are USE XML, USE EJB, USE WEBSERVICES, …

USING UML can be very efficient in situations such as the INVERTED POWERPOINT ARCHITECTURE, because it makes the presentation look even more impressive.

Last but not least, if you don't have these tools & methodologies people available, there fortunately is some help available because there are several authors who have written down many of these aspects in the form of development processes and methodologies. If you are unexperienced and don't know how to successfully use HALF-XP, follow a PROCESS BY THE BOOK, implementing *all* techniques and artifacts proposed. There is a temptation to omit optional features prescribed in the book, but this temptation should be resisted as you are trying to get the most value out of the method, and omitting anything will only dilute that.

PROCESS BY THE BOOK

Context: You need to run a project and you don't have a TOOLS AND METHODOLOGIES DEPARTMENT TO TRUST.

Problem: How do you know which process to use, which practices to implement and which artifacts to produce?

Solution: Take a book on one of the more heavy-weight processes and follow every single instruction outlines in the book. Implement all artifacts and practices exactly as described there.

Now go and start your project! You're well prepared!

Acknowledgements

First, I'd like to thank all those people who participated in all those projects from which I "mined" these patterns. You've done a great job!

Then I'd like to thank Kevlin Henney, who played the shepherd at EuroPLoP 2002. He contributed many useful comments, and also spawned the idea for the PLUG-AND-PLAY PROGRAMMER pattern. Being a native speaker, he also helped to polish language issues to make it even more ironic and cynic.

Also, I'd like to thank EuroPLoP 2002's writer's workshop D for their useful comments, suggestions and discussions, as well as for their very positive and encouraging feedback.

I also would like to thank Jutta Eckstein, who gave me good advice on how to formulate some things in order to make sure that those people who were on these project don't feel offended of what I wrote. Manuela Nagel gave me some good comments from the perspective of a not-so-cynic person, and Torsten Holmer hinted at the term *Publish and Forget*.

References

[AC97] Cockburn, Surviving OO Projects, Addison-Wesley 1997

[AC01] Alistair Cockburn; *Agile Software Development*; Addison-Wesley 2001

[AC02] Alistair Cockburn; *Some Article I need to find out details about*; Look at http://www.aanpo.org/articles/articles/ACcitj0102.pdf

[FB95] Brooks, The Mythical Man Month, 1995

[KH02] Kevlin Henney; *The Imperial Clothing Crisis*; http://www.curbralan.com

[OS01] Olson, Stimmel, *The Manager Pool,* Addison-Wesley 2001