Introduction

"In the many years I have been working as a consultant, I've had to convince clients of a variety of things. That they should not be using interfaces for data conversion, that it's best to have a team of solution architects with design approval authority, that they need a vanilla environment to retest errors to be sure the error wasn't caused by a modification, that most modifications were in fact unnecessary, that the project team needs training prior to starting the project, that you should have a batch owner, etc. Large things and small things. Sometimes I succeeded. Most of the time actually. But usually I had the client convinced after it was too late to change anything. Right after them asking me "why didn't you tell us!"."

An anonymous consultant

"Throughout the implementation of our new ERP system, I had consultants politely mentioning, emailing, strongly suggesting or even yelling advice at me on just about anything. My team has a solution to a problem, some consultant walks in and tells me "this is not how it should be done because there'll be trouble" or "this is not proper architecture" or "you can't use an interface for data conversion, everybody knows that". Then when I ask for some arguments to back up those statements I get a 200 slides long Power Point presentation that needs an archeologist to unearth a single sentence that makes sense to me. Meanwhile the project misses deadlines and has to deal with budget overruns."

An anonymous client project manager

It is sometimes tempting to think that it is due to people's ignorance when good advice gets ignored. However, that reasoning doesn't hold. The client has a lot at stake on ERP implementations and not only wants but needs a successful project. Consultants love to add successful projects to their resume, and if they're the right sort, they do take some pride in what they do. So let's take a positive attitude and assume everyone involved wants a smooth implementation that delivers an ERP solution on time and under budget. Why then wasn't the advice convincing enough? Should the information be provided in a different way? Why weren't the benefits of the advice obvious?

1 You should divide projects into phases

An introduction to Project Phases

Summary

Most implementation projects will follow a similar structure. The naming of project phases may differ from project to project, the activities that should take place during those phases do not. It is important to understand how projects are structured, and what goes on in each phase of a project.

Introduction

The word project comes from the Latin word projectum from the Latin verb proicere, “before an action” which in turn comes from pro-, which denotes precedence, something that comes before something else in time (paralleling the Greek ϖ) and iacere, “to do”. The word “project” thus actually originally meant “before an action”. When the English language initially adopted the word, it referred to a plan of something, not to the act of actually carrying this plan out.[1]
If we take a step back to ERP [2] implementations, projects are there to prepare companies for using this new ERP system, and configure the system to fit the company’s needs.
Athens wasn’t build in a day, and neither are ERP systems: an average ERP implementation

1 http://en.wikipedia.org/wiki/Project#Overview

2 Enterprise resource planning (ERP) systems integrate internal and external management of information across an entire organisation—embracing finance/accounting, manufacturing, sales and service, customer relationship management, etc. ERP systems automate this activity with an integrated software application. ERP facilitates information flow between all business functions inside the organisation, and manages connections to outside stakeholders. http://en.wikipedia.org/wiki/Enterprise_resource_planning

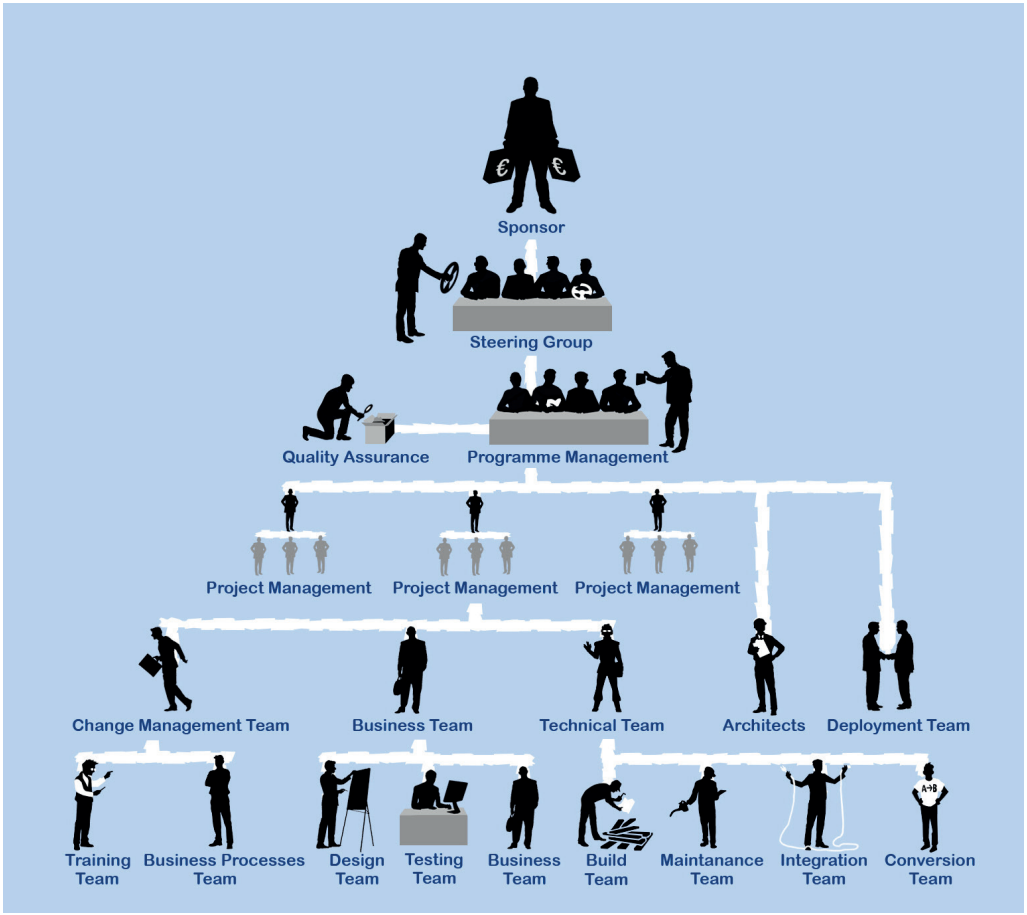

Figure 1: Project organisational hierarchy

Let's start at the top.

The Programme Sponsor

Every large scale implementation will require a Programme Sponsor. This will not only prevent parts of the company running unsupported projects on their own, but is also part of the programme governance model. Someone needs to be ultimately responsible for the success or failure of the implementation. This person is also responsible for the programme budget. More often than not this person is the client CIO[1].

1 Chief Information Officer

The Steering Group

Is usually chaired by the Programme Sponsor. The Steering Group (a.k.a. Steering Committee) consists of line management representatives of all affected departments of the ERP implementation, Programme Management and representatives from the software vendor and implementation partner(s). This Steering Group ensures the programme stays within budget, makes decisions regarding programme changes with a big impact, initiates quality reviews and verifies that the programme deliverables are in line with the business requirements. The budget responsibility also means that any budget change is brought to this group, such as change requests and additional resource requirements that increase programme costs.

Programme Management

Manages the complete ERP implementation. The programme consists of multiple projects. Each project could for example focus on the implementation of a specific application or business function (e.g. “import”, “stores” etc.). Programme management focuses on the delivery of all projects and makes sure they are aligned in terms of deployment strategy and dates, integration and end to end business processes. They report on progress to the steering group. The role is often fulfilled by an external consultant, but it is recommended to at least have one internal programme manager.

The Quality Assurance Team

Reports to programme management or to the steering group on the quality of the deliverables of the programme and the individual projects. They review a subset of all deliverables (design documents, test plans, conversion strategies etc.), and the methodology used on the implementations. A QA role is usually performed by an experienced (external) solution architect, supported by an internal auditor and the team of architects.

The Architect Team

Has a similar role to Quality Assurance, but is more practical in providing cross-project support on integration and design. The architects review all design documents, plans and strategies (e.g. on testing, integration, conversion etc.), and ensure that the integrity of designs is maintained though all project phases. They also provide advice on “best practice” (also more correctly known as “leading practice”) and

Why

You should involve your best people in the project

1 The company's own IT Maintenance and Support team should supply most of the resources for the programme's Maintenance Team.

BUT... The Maintenance and Support Team is already busy supporting legacy systems in production, they don't have the time to participate in a project. A handover at the end of the project is good enough.

SO... *Using internal IT resources will ensure they learn to support the new systems, and they will provide the knowledge of the existing systems. To help build experience they should also investigate and support fixing of all software issues that arise during the project (e.g. during Test Phase).*

2 No project should cut costs by getting rid of the experienced subject matter experts.

BUT... SMEs are expensive. Once they've trained our staff (or a cheaper 3rd party implementor) they are no longer required.

SO... *Without experienced SMEs you are less likely to run a successful project, as it is the one role who knows the systems that are being implemented inside out. This knowledge is not only required during the Analysis and Design Phases, it is also vital for a successful Build and Test Phase.*

3 Listen to the advice your external consultants and architects are giving you.

BUT... "This is not how we work here".

SO... *This should be obvious. When asked everyone will say the follow good advice. The trick lies in knowing when advice you've been given is good advice. Just remember that your external team of SMEs and architects have implemented this new system far more often than you have.*

4 Interview consultants before they arrive on site, ideally supported by a trusted advisor. Review their LinkedIn page. Ask around. This may sound like too much work for a large scale project, but trust me, it's worth it.

BUT... "No time for this. I wouldn't know what questions to ask anyway".

SO... *It is important to assess if the consultant has the product or industry knowledge you've asked for. It will also make sure that the implementor and software vendor you are working with will offer better quality consultants.*

5 Plan for expected resource changes: people who can't or won't return to their old job, and consider new roles that need to be fulfilled.

BUT... "We'll get there when we get there". Planning for this takes close cooperation with HR, which is a line of communication often ignored on projects.

SO... *The sooner you have a view of future internal resource requirements, the better. Work instructions may need to be written by the project for these new roles. Staff may be more motivated when they understand what options they have for career moves after the project.*

5 You shouldn't reimplement legacy

Defining the scope, business case and design approach

Summary

When implementing a new ERP system, there is a strong tendency on all projects to make the new system fit into the old way of working. As a result, the new software resembles current systems a lot and its potential is not fully utilised. Ways of mitigating this risk can be found in defining a detailed business case and scope and by having a design approach that finds the right balance between the best and the achievable solutions.

Introduction

"Colleagues! Welcome to the kick-off session for the implementation project of our new ERP solution, which we named "The Future"[1]. We are glad to see so many of you, full of anticipation of a marvellous journey ahead of us! And we know why we are doing it: to ensure growth opportunities for our company, to streamline our processes, and further expand our presence in region X! Our focus will be on changing systems for HQ initially, later projects will implement new

1 Projects, in my experience, are usually given one of the following types of names:
- Positive, mission-statement-like project names about Change and Bright Futures, which can include words like evolution, grow, future etc.
- Acronyms. These often lead to misspelled country or animal names. "Realising A Better Information Technology", "or GRowth through a new Evolving Architecture Collaboration Enterprise" are some fake examples that are no more far fetched than some real world examples I'd rather not mention.
- Ominous names. Be careful when using this type, as the name can be a foreboding of an unsuccessful project. One client I worked at used venomous snakes as names for all its projects. They didn't end well.

systems for our stores and warehouses. And remember: vanilla, vanilla, vanilla! Don't change the new applications unless it touches one of our defining processes and it is an absolute must have!"

Silence, then applause...

The above excerpt of a kick off speech is a variation of speeches I've heard many times. Some are better than others, and they usually are meant to inspire the business team (the external team doesn't need motivation, though they may have different reasons for being there), and to explain the Why, the How and the What.

The Why?

Question should be answered with the reasons for running this project. This depends greatly per project, but usually falls within one of two categories: to reduce total cost of ownership (IT or Finance driven) or to increase scalability and flexibility (Business driven).

The How?

Question addresses the project organisation (who does what), the ground rules (vanilla!), and the project plan (phases, milestone plan).

The What?

Question should be answered with a list of systems that will be replaced, the processes that will be affected, etc. To put it in one word: Scope.

For each of the above, the initial draft may be easy to accomplish. Adding details and guidelines, and ensuring these are followed throughout the project; that's where the challenge is. Without addressing these details, and knowing how to enforce them, you will end up implementing a new system that looks and acts remarkably like your current legacy systems. For example, stating that the new systems should be used as vanilla as possible, won't do the trick. Very tight control on software changes is needed because the business will deem most change requests as "must haves". And most of these will try to make the new system act like the old one. Another example is about the business case: projects that started to reduce total cost of ownership may fail because they don't deliver enough business benefit. And business driven projects usually lack IT support. But I'm getting ahead of myself. This chapter will address some of the details of:

Business Case ("why"):

The reasons for starting a project, and the ability to communicate these greatly influences the outcome.

Design Approach ("how"):

The methods used for designing new processes define how much freedom the project has in finding the best possible solutions, which may be very different from legacy systems. Not enough freedom is bad, but so is too much of it.

Scope and Mandate ("what"):

This determines the tools in the toolbox, and who is allowed or forced to use them. What room do project members and the project have to change processes.

Business Case

"We had a big delay in our project, as the business didn't want to sign off on the final design of the new ERP system. The reason for starting the new project was to replace the old mainframe systems that lost support years ago, and we needed to reduce the maintenance cost quickly. The business was fine with this as long as they didn't lose any functionality, and we didn't disrupt their activities too much during the implementation. We did all that, but once they fully grasped what the new project could do for them, we were already two-thirds in, and we were ready for testing. That's when we changed the scope, but we forgot to update the business case."

The above quotation comes from a project where the business case was build on reducing total cost of ownership (TCO). This happens a lot, as it is easier to estimate the TCO than it is to estimate business benefit. Cost of ownership is based on e.g. license costs and the costs for IT support (which can be expressed in number of people needed to support and maintain the solution). If you compare this figure to the current total cost of ownership, and the difference is enough to fund a project to implement a new system, than that provides you with all the business case you need to support the new implementation. Business benefits don't seem to be required. Even if business benefit is included, it is likely the benefits are described in terms of efficiency, reduced headcount etc. More thought-through business cases may include the benefits of reduced inventory, reduced buying costs, or even include an estimated benefit of e.g. having scalable systems or increased data mining capabilities. In my experience, the more detail the better, as long as it is quantifiable.

A business case is used to ensure the project funding. The project basically "asks" the company for a loan to cover the license and implementation costs, and promises a positive return on investment (ROI); plan and proof of performance being the business case. This is why during the implementation the steering group, that governs the project, uses the business case to measure (potential) success and failure of the implementation. All questions around budget changes, scope

A brief summary of the chapters

Chapter 1 You should divide projects into phases,
provides an introduction to project phases. Even though the naming of project phases may differ from project to project, the activities that should take place during those phases do not. It is important to understand how projects are structured, and what goes on in each phase of a project.

Chapter 2, You should involve your best people in projects,
discusses roles and teams on ERP implementations, focussing on internal business resources, who play the most important roles on projects. They provide the project with the knowledge it needs to implement new processes and a working system, and they are internal sellers of the changes the organisation is being put through. It is important to understand at the start of a project where they are needed, and to use as many of them as possible.

Chapter 3, You should treat external consultants like one of your own,
explains how to get the most out of external consultants. Working with external consultants is unavoidable on a large ERP implementation. If it is your first time, it may take some getting used to. And even for the people out there experienced in dealing with them, this chapter contains a number of tips to help you maximise their output.

Chapter 4, You should use your own implementation methodology,
provides tips for creating a methodology that fits your needs. Using an implementation methodology is vital to the success of an ERP implementation. It is more than a collection of templates, it should describe how your company "does things". To make the methodology your own, this chapter pays special attention to using target application landscapes, non-functional requirements and architectural guidelines.

Chapter 5, You shouldn't reimplement legacy,
is all about defining scope, business cases and design approaches. When implementing a new ERP system, there is a strong tendency on all projects to make the new system fit into the old way of working. As a result, the new software resembles current systems a lot and its potential is not fully utilised. This chapter discusses ways of mitigating this risk.

Chapter 6, You should build a Knowledge Base,
is all about keeping application and process information organised and accessible. Before an implementation starts a knowledge base process should be established and communicated to ensure all knowledge that floats around on projects gets captured, and all project deliverables are made to fit a company's knowledge base.

Chapter 7, You should have a vanilla environment,
is about defect management, defect ownership and vanilla environments. During test phases many issues will arise with custom or base software, integrations, conversions, etc. Streamlined issue triage is one aspect that is vital to the success of the test phase. Another aspect to this is SR ownership: companies should own their SRs as soon as possible as it builds knowledge and experience. Issue resolution itself is greatly helped by having a vanilla environment, especially where modifications are involved.

Chapter 8, You should patch,
explains why you should define a patching and upgrade strategy. Often companies forget to plan time and budget during the implementation to allow for patch upgrades. When the project runs for a long time, software issues may accumulate and require a patch upgrade. The project plan should have enough time allocated for retrofitting modifications, potential tech stack updates, regression testing, conversion and configuration updates. Having an upgrade strategy will help the implementation team focus on the (re)usability of all deliverables.

Chapter 9, You shouldn't modify, unless you really have to,
clarifies why modifications should be avoided as much as possible. Organisations often try to recreate legacy in their new system, because thinking outside the box is difficult. All business processes must be reviewed end-to-end, which will enable more opportunities to push out modifications. If modifications are inevitable, make them "bolt-on" as much as possible to stay on the upgrade path.

Chapter 10, You should have a proper design and review process,
will help you design your design process. The quality of design documents, whether they are high level designs, functional designs or technical designs, increases dramatically when a thought through and documented design and review process is established. This does not mean designing and reviewing will take up more project time and budget: it will save you time.

Chapter 11, You should assign data owners,
explains why data ownership is key to good data. Data owners come in two varieties: systems and people. Every piece of data should have a master, i.e. the single system in which this data is maintained and sourced from to other systems. And every piece of data, or rather data entity, should have an owner; a person responsible for maintaining data integrity, the data model, and the integration of the data across systems.

Chapter 12, You should clean your data,
discusses why data cleansing, conversion testing and conversion sign off is important. Data conversion is often underestimated by projects, in terms of complexity and in terms of effort. Conversion of data should be an automated, repeatable process that is executed in a dedicated conversion environment.

Chapter 13, You should support your people after go-live,
talks about the importance of post go-live support, and how to best organise it.