Why Large IT Projects Fail

Peter Henderson
p.henderson@ecs.soton.ac.uk

School of Electronics and Computer Science
University of Southampton, SO17 1BJ, UK

May 2006 (revised August 2010)

Abstract

Large IT projects may not deliver what they promise. Often they are late. Often they are over budget. Often what is eventually delivered is not either what was originally specified nor what is actually needed. There are many reasons why this happens. We discuss some of those reasons here and argue that many of them are a consequence of failure to specify requirements anything like adequately. We introduce the notion of requirements drift as a root cause of failure and show, by the use of models, that the often quite large discrepancies between expectations and reality can be the consequence of quite modest amounts of requirements drift.

This preprint is available at http://www.ecs.soton.ac.uk/˜ph/LargeIT.pdf

Keywords: Large IT Projects, Requirements, Failure, Open Systems, Complexity, Models

Introduction

Large IT projects are those that involve hundreds of software engineers and require many years to complete. Such projects often fail. Failure can be in various forms. Sometimes the delivered system fails to provide significant requirements to the standard of quality which makes it possible to actually deploy the system. Sometimes the anticipated cost for the project is exceeded by a large factor. Sometimes the anticipated delivery date is exceeded by a large factor. Often, failure involves a combination of all three of these effects.

The main problem is to determine the causes of such failures, since this must precede the recommendations for mitigating their consequences. We will enumerate the accepted reasons for large project failure and show that an underlying cause of all of them is what we shall call requirements drift.

The accepted reasons why large IT projects fail are many and various, but include all the following

1. There is too much optimism as to the potential benefits of IT and as to the cost of delivering those benefits

2. There is too little investment at the beginning.

3. Sufficient investment is made eventually, but is too late.

4. There is not enough technical know-how in the project team.

5. There are opposing Human Factors aspects -including insufficient consideration of teams, project management and risks.

6. The project does not plan incremental roll-out and fails to anticipate the effect of big-bang on the organisation.

7. The project tries to match IT to the existing Business Processes of the organisation, rather than mandating that the Business Processes change to match those implicit in the IT.

8. Initial underinvestment leads to an investment legacy, where the project has invested in bad decisions and doesn't have the courage to retreat.

9. There are many management disaster scenarios such as

   (a) parent/child governance scenario -where authority within the project remains with senior management in the customer.

   (b) the enthusiastic builder scenario -where the supplier has a vested interest in prolonging the project since payment is related to time-on-the-job rather than delivery of results.

10. And of course, because of insufficient planning and simply because projects take a long time from inception to completion, there is a requirements explosion during the lifetime of the project which means that what is eventually required is significantly different from what was originally anticipated

In fact, because of the impossibility of accurately capturing requirements (where supplier and customer absolutely have the same understanding) before a project begins, and because an organisation learns over time, requirements are subject to constant change, a phenomenon we call requirements drift. Each of the reasons listed above can be shown to have an underlying cause which is explained by requirements drift.

We have built a series of models of large projects, based on our understanding of how real projects perform [Chatters 1998], and used these models to explain how failure could be anticipated if requirements drift were considered from the outset and throughout. Our results demonstrate the quite remarkable butterfly effect of modest changes in the rate of drift. Small changes in that can lead to significant changes in time, cost and quality, the principal measures of success.

Hence, if requirements drift could be anticipated, if it could be measured and if it could be planned for, realistic expectations as regards success could be set and potentially achieved.

1.1 Outline of Paper

First we characterise what we mean by large IT projects and what it means for them to fail. Then we put some flesh on the bones of the ten reasons listed above for project failure. These reasons are, of course, interrelated, so there is no best order in which to present them, since whatever we do we have to make forward reference to reasons that we have not yet elaborated. It is a conjecture of this paper that everything is related to the (in)ability to keep control of requirements, so every reason will make forward reference to that.

Next we introduce the notion of requirements drift which captures the observation that as a project proceeds, the requirements change. Many of these changes are actual, but many are simply a consequence of the organisation learning more about what the original requirements actually involved. In this section we argue that the ten reasons previously discussed can all be seen as an aspect of requirements drift, or can be shown to be exacerbated by requirements drift.

Then we introduce a model that we can use to explain the relationship between requirements drift and project metrics. We show, by the use of this model, the effect of modest requirements drift on project cost and schedule and on the quality of the delivered product.

Finally we discuss how this insight can be used to better control projects and to mitigate the consequences of project failure.

2 The Characteristics of Large IT Projects and their Failures

2.1 The Characteristics of Large IT Projects

There are a number of ways in which a large project is measured. Perhaps the most obvious is the cost. However, this measure cannot be considered in isolation. For example, one way to lower the cost of a product, is simply to deliver something of lower quality. Or, what amounts to the same thing, to deliver it too early. Consequently, when discussing large projects we need to consider a number of dimensions. Here, we distinguish Size, Quality, Cost, Schedule and Complexity. Before we discuss their relationship, let us first define them.

2.1.1 Size

Large IT projects are dominated by the need to deliver software, in all its many diverse forms. Normally, contemporary projects involve a much greater proportion of existing code, that simply needs to be configured, than they do original code. The software engineers spent their time modifying some of this code, in particular setting parameters and writing scripts that will specialise the existing code to the specific requirements of this project. Thus, the quantity of delivered software may be measured in millions of lines of code, but only a few thousand of those lines will be new. Nevertheless, the larger figure is an essential measure of size, since in many repects it quantifies the amount of work that the engineers need to do in order to understand the functionality delivered by a given component before they can accurately specialise it. There are established means of calculating how much work will be involved in producing code [Boehm 1981] which have been refined in recent years to handle this contemporary case of building largely from existing components [Boehm 2000]. These methods require good historical data from similar projects, ideally those delivered by the same supplier. This data is seldom available. Other measures of size are necessary.

Once a project is complete, a reasonable measure would be the number of (hundreds of) engineer-years required to complete it. Clearly, projects continually try to estimate this figure from their inception, since it is a major component of the delivery cost. Where software is concerned, such measures are often wildly inaccurate. The Mythical Man-Month [Brookes 1979] tells us that, for many reasons, the delivery of a software product can require up to ten times the effort of producing a proof-of-concept prototype. This figure is seldom believed by managers and is frequently the reason that initial estimates of cost are substantially too low. This can be thought of as an inability to estimate the size of the product accurately.

One conjecture is that the best measure of size for a project is to count the number of requirements. Clearly the more requirements there are, the more work there will be to do, even if it is only to confirm that an existing component does indeed deliver a given requirement. Of course, this begs the question of how big is a single requirement, and even whether that question is well-formed. But, assuming we can enumerate all requirements, collect them in a document and break them down into unit requirements (say, for example, something anticipated to require one engineer-month), then we would have a reasonable prediction of the size of a project. After all, this is exactly what we do in project management when we construct a Gantt chart and agree a delivery schedule.

2.1.2 Quality

Now we come to the measure of what it means for a delivered IT project to be of acceptable quality. When project progress is reported informally (in the press, for instance) it is often the case that the perceived end-user quality is unacceptable. The system may not deliver the anticipated functionality, it may be very slow or it may be very difficult to use. It may even still break in the users’ hands, where residual bugs expose themselves too readily.

Considering only this extrinsic quality, we can imagine the end-user sitting down with a list of requirements, and analysing the system to see which of those requirements are delivered to their satisfaction. If we imagine them ticking those requirements that they find acceptable, then we would eventually have a crude measure of (their perception) of the delivered quality. Optimistically we might hope that they had ticked 90%, 99% or even 99.9% of those requirements. These would be significantly different measures of quality, perhaps only the last of those actually being acceptable in a realistic application and often higher levels of quality are expected, especially in a safety-critical context. Moreover the cost of going from 90% to 99% can exceed the cost of getting to 90% in the first place, and then getting to 99.9% can, in turn, exceed that. These diminishing returns are issues that we will return to later.

And of course, there is intrinsic quality. This is a matter of how well the product is designed and built. This will have major effects on the lifetime costs (total-cost-of ownership). We will argue later that extrinsic and intrinsic quality are closely correlated.

2.1.3 Cost

Large projects cost large amounts of money. The recent project “National Programme for IT” in the UK NHS [NHS 2005] was originally costed at around £6,000m, whereas a recent official report estimates that the eventual cost may be double that and some reporters have even predicted £20,000m before delivery is as originally planned [NAO 2006]. These are very large numbers, but the effect is not uncommon, an initial estimate that doubles, then trebles and worse. Many large public works go over budget by similar factors, For example, in his report on the inquiry into the cost of the Scottish Parliament building, Lord Fraser states the inquiry was set up to evaluate a project “which is now two and a half years behind schedule with costs running approximately ten times more than the original estimate of 40m.” [Holyrood 2004]. While going over budget is not unique to the IT industry, it is increasingly familiar in respect of the size of the discrepancy between initial estimate and ultimate cost [Humphrey 2004].

In the case of large IT projects, the cost is largely for staff. The procurement aspect of large projects is usually dominated by the cost of employing sufficient staff. This in turn is estimated on the basis of perceived size and complexity of the building task.

Large projects are however normally the subject of competitive tendering and the initial cost proposals are clearly dominated by the need to win the bid. Purchasers of large IT will try to mitigate their risk by tying the supplier in to some punitive form of damages but often this just causes one or more of the consortium to collapse. The customer is left with a legacy investment and a partially completed project. The money has gone. The choices are stark. Invest more to continue (with new suppliers) or go without the solution. Effectively the project starts over. A cynic might say that large projects require two or three such restarts until the organisation learns sufficiently about the problem in hand, until more realistic objectives are established [Cambridge 2001].

2.1.4 Schedule

Delivery on time is also unusual for large IT projects. Many of the steps in a development process are necessarily sequential. Software can’t be tested until it is built. Integrated systems can’t be built and tested until their components are available. Non-functional requirements (performance, human factors etc) can’t be tested until a realistic subset of the functionality is available and a large scale evaluation can be established.

The project will anticipate that many of these things can be overlapped. In theory, that will be true. In practice, it will be difficult to achieve. Often we have a project that has been planned with dependencies on a number of key components (e.g. promised functionality in a new version of an operating system). Late arrival of the key components result in delays and reworking, adding to schedule slippage and overall cost.

Again, the Mythical Man-Month [Brookes 1979] tells us that person-months are not subject to arithmetic laws. A task may well require 12 person-months, but it may (for example) be that this necessarily is in the form of 2 persons for 6 months. Adding a third person will not reduce the schedule. Three people still require 6 months. Indeed, they may require longer, if the added person needs to be trained, since this calls time away from the initial two. This is sometimes quoted as “adding more people to a late project makes it later”.

These phenomena are well known in the industry. They are consistently overlooked. Because they are not Laws in the sense of Laws of Physics [Henderson 1998], manager after manager will convince themselves that they do not apply in their particular circumstances.

2.1.5 Complexity

Last, we must consider the inherent complexity of large IT projects. Clearly, this varies from project to project, but there is no doubt that software in general can be very complex (in both the formal and informal senses) [Axelrod 99, Kauffman 2000, Chapman 2002, BCS 2004, Xia 2004].

Complexity usually means that the separability of the components of a product are severely constrained. Changes to one component induce changes in many others. This is referred to as being tightly-coupled. A good design will seek loose-coupling or near-independence between components. However, good design isn't often achievable in large projects -because of their inherent complexity; a cyclic problem if ever there was a one. Few project managers have the courage to invest more in design in order to save later on building.

There is some threshold of connectedness above which the complexity of a large IT system becomes intractable, not unlike the thresholds discovered in combinatorial problems [Hayes 2003]. Below this threshold, the work required to solve a problem is effectively linear in some aspect of the size of the problem. But above the threshold, work increases exponentially (or worse) and soon problems just a little above the threshold are, for all practical purposes, unsolvable.

Complexity is often exaccerbated within the project, where building loosely-coupled solutions takes longer than building tightly-coupled ones. The project looks only at the short-term costs and not at the total-cost-of-ownership. A too-quickly developed solution will usually be more tightly-coupled than it could be, and that will lead to problems with internal complexity much later.

2.1.6 On the relationship between Size, Quality, Cost, Schedule and Complexity

Clearly these five aspects of largeness in a large project are themselves related. We have referred already to some of these relationships, for example larger projects will cost more. In fact we even suggested that cost was a reasonable measure of (post facto) size. We will not try to formalise the relationship, but an informal discussion is warranted.

Figure 1 is a diagrammatic representation of the relationship between these five values and is based on a model that we have used in earlier experiments [Chatters 1998]. The box in the centre of the diagram is intended to denote the development process (think of it as a factory) which takes in components of a certain Quality and builds and outputs a product of a certain Quality. The dominant measures of the size of the task at hand are the perceived Size and Complexity of the product. By doing Work on the product (i.e. designing, building and testing it) we can affect the Quality of the output. We can also affect the delivery Schedule.

For example, if wereducethe Work rate below somemaximum, wewould expect theScheduleto increase. However, trying to go the other way, we discover there are limits to how much the schedule can be reduced by increasing the Work rate. Person-months do not obey the laws of arithmetic.

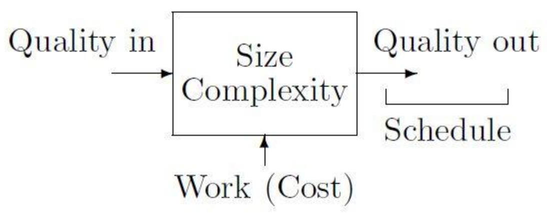

Figure 1: Overview of the Development Process, with Measures

Similarly, if the input Quality of basic components from suppliers is high, we would expect to achieve a target output Quality for less Work. Here, we align Work with Cost, on the basis that person-years is the dominant cost in large projects.

In [Chatters 1998] we showed that output Quality is related to Work by a law of diminishing returns; that as the project gets bigger the amount of delivered Quality per unit of Work reduces. This rather obvious property was supported in that paper by some rudimentary data, but data of that sort is very difficult to acquire because it can be very sensitive. We found that the data we did collect was rather unreliable and so it was not possible, at that time, to establish the relationship precisely.

2.2 What Failure Means

Given these five Measures we can state what it means for a large IT project to be perceived as failing, or to have failed if it has been abandoned. Since it is often the case that large IT projects are perceived as failing long before they are finished, we address the symptoms in the order in which they are usually reported. First we discover that the project is going Over Budget, then we discover that it is Over Time and finally we discover that the customer sees it has having Low Quality.

2.2.1 Over Budget

Many IT projects cost a little more than originally budgeted. The characteristic of large IT projects seems to be that they go over budget by a very large factor. Then, the fact that their budget is already very high, means that the additional cost will be newsworthy.

Since this cost over-run will normally happen before the original schedule has run its course, it will normally be the need for the suppliers to renegotiate their contract that is the first indication that a project is failing. As we shall see, it is the realisation that the requirements apparently agreed in the initial contract cannot be delivered for the initially quoted price that kicks off this renegotiation.

2.2.2 Over Time

Soon after the realisation that the initial budget is inadequate is the realisation that the initial schedule is also unrealistic. These aspects are related. The project has proven to be much bigger than anticipated, so more Work is required (i.e. more Cost) and it will take longer, because we cannot trade person years (except within fairly tight limits).

2.2.3 Low Quality

Finally, when a first product release is eventually available for end-user evaluation, it is reported that the product does not fulfil the requirements that the users now have for it. The Gedanken-Experiment that we discussed earlier, of the end-user apparently ticking-off requirements on a long list and then measuring Quality as the number of ticks that the product receives, gives us a crude measure. Often this measure is low when a large IT project is delivered. The reasons are many and various, as we shall elaborate in the next section. But primary among them is that the requirements as understood now by the end-user are some way from those that were agreed at the outset. Formally, the difference may be small. In practice, it may be all that matters when it comes to reporting on the perceived quality of the eventual product.

2.2.4 Comments on Perception of Failure

As we discussed above, the three perceptions of failure, respectively being too expensive, being too late or being of poor quality are interrelated. Often we have all three, but even having one of them would be considered a failure.

We will argue that it is an inadequate attention to requirements that leads to failure. There are two aspects of this. Of course, over the lifetime of a long-running project, the requirements will change. Perhaps more significantly, the initial requirements will also “change” in the sense that, as they become more fully understood on the part of both supplier and customer, it is increasingly realised that the initial agreement was based on “weak” mutual understanding. What was perceived as agreement, is now seen as a basis for conflict. The supplier wants to deliver what they originally undertook to deliver, the customer wants that too, but disagrees as to the details of that initial agreement. The customer now wants something additional however, something more refined or more elaborate that the initially agreed requirements. This puts the customer in a weak position to negotiate a compromise.

2.3 Reasons why Large Projects fail

In the overview, we briefly described ten reasons why large IT project fail. Now we have some vocabulary with which to discuss those reasons. The order in which they are addressed is rather arbitrary. Since they are all interrelated, no order is perfect. Eventually we will show that the root cause is our failure to manage the requirements process adequately and, in particular, that requirements drift goes largely unnoticed.

2.3.1 Optimism

There is too much optimism as to the potential benefits of IT and as to the cost of delivering those benefits.

Both customer and supplier expect more than is realistically possible. The customer expects that the product will solve all their problems. The supplier sells that dream. Both may well believe that the agreed requirements can be delivered for the agreed cost and that those agreed requirements will actually meet the customer's need. They are too optimistic.

2.3.2 Investment

There is too little investment at the beginning.

This optimism leads to a confidence in the estimated cost which results in an initial investment that is far too small. The investment is likely to be related to a spending forecast that anticipates an unfaultering project plan. Something like 25% is added to cope with "unforseen" circumstances.

The reality is that there was never really any prospect of delivering the project for 200% of the estimate, let alone 125%. The Size and Complexity have been underestimated. The Quality of existing components (in the sense of their ability to nearly match some of the requirements) has been misjudged, and hence the rework necessary has been underestimated. This is true even when Open Systems components implementing open standards have been used. The work required to actually understand how these components work, and in particular how they interoperate with others, is usually much more difficult than is anticipated.

2.3.3 Procrastination

Sufficient investment is made eventually, but is too late.

So reality dawns. More investment is needed. This is the first of what might eventually be a number of "stumbles". Cynically, we might say that, for all large projects, one or two stumbles are required as the organisation (the suppier and the customer) learn about the complexity of the task they have embarked upon.

However, had the initial investment been larger, the project would probably not be so far behind. This is something that our models show. Anticipating a more distant objective can accelerate progress. So having an initial establishment that makes more distant objectives possible, means that the eventual cost of delivering a given function to a given quality will actually be less than that which will be achieved with insufficient initial investment, even if that is eventually "topped-up" to an adequate amount.

2.3.4 Technical Know-how

There is not enough technical know-how in the project team.

Of course, making sufficient investment probably assumes that the staff that the supplier uses for the project are of sufficient technical quality at the outset. Optimism sets in here too. We all assume that we are able to achieve more in a given time than in fact we are capable of achieving. We usually over-estimate our own technical ability. Experienced project managers allow for this. There are generic technical skills which can be established and allowed for at the outset. But there are specific technical skills, knowing the capabilities of the components we are building from, knowing the application domain sufficiently well that the requirements can be understood, which are likely to be misjudged. We are optimistic. We expect the supplied components to be “better” than they eventually prove to be. We believe we understand the customers’ requirements at least as well as the customer understands them. The reality is that the components are not nearly as fit-for-purpose as we anticipate (even when they adhere to open standards) and what we understand about the requirements is far from what the customer intends, even when they reach agreement andsignthe contract.

2.3.5 Human Factors

There are opposing Human Factors aspects -including insufficient consideration of teams, project management and risks.

This is a large topic, and many writers will suggest that solving project management problems is the single most important aspect of delivering Quality on Schedule and for an agreed Cost [Brookes 1979, Petroski 1992, Grey 1995, BCS 2004, ACM, Neumann 1995]. These claims are of course valid. A poorly managed project, without an appropriately organised team structure and without continual reassessment of the risks ahead, will not deliver on any of the three measures of success. However, many large IT projects are managed correctly in these respects and yet they still fail [Intellect 2003, Sauer 2003, Xia 2004, Humphrey 2004]. So, while getting the “people” issue sorted is necessary, it is not sufficient.

2.3.6 Incremental Delivery

The project does not plan incremental roll-out and fails to anticipate the effect of big-bang on the organisation.

A project that introduces new technology into an organisation incrementally is more likely to eventually be seen as successful. There are two reasons for this. The first is simply that each increment is the product of a smaller project. The second is that, where there are problems that are related to misunderstandings between supplier and customer, these problems will be encountered earlier and have smaller consequences.

A component-based solution, using commodity off-the-shelf components (COTS) can make incremental delivery possible, especially where these components adhere to open standards as expected in an Open Systems approach. There are many benefits to introducing these new components incrementally, not least that problems are encountered early, even if it is only problems of perception -understanding what the components actually do and how they actually match (or fail to match) the customers requirements.

2.3.7 Inertia

The project tries to match IT to the existing Business Processes of the organisation rather than mandating that the Business Processes change to match those implicit in the IT.

IT offers a business more efficient and more effective ways of running their business processes. But where, as is usual today, the project to introduce new IT starts with an existing software solution
(e.g for ERP or CRM) much of the engineering that is involved is specialising the IT to the business processes of the customer. Where the customer insists on continuing to run the business as before, these engineering changes can be overwhelming. One suspects that for many projects they are actually impossible.

The IT packages that are built upon will have built-in generic business process models. Where the customer is willing and able to change their business practices to more nearly match those generic processes, then the amount and complexity of the engineering that will be required is reduced, often substantially.

The issue here is really that two inconsistent models are being reconciled. If both are subjected to modest change, rather than one being forced to fit the other, the overall complexity of the problem to be solved will be minimised. This minimisation may be essential to success. The organisational learning required (on the part of the project, but in particular on the part of the customer) before this issue can be addressed may be the reason that large IT projects often undergo two or more "restarts" (or "stumbles"). The adjustments made at these events are often to persuade the customer to change their business processes to match those more readily supported by the IT. If this had been accepted as necessary at the outset, then considerable cost and delay might have been avoided. This is particularly true for Open Systems, where maintaining the integrity of the interfaces, necessary for future evolution, often requires accepting that it is better to move the business processes to match the IT, than the other way around.

2.3.8 Legacy

Initial underinvestment leads to an investment legacy, where the project has invested in bad decisions and doesn't have the courage to retreat.

An issue that arises as a consequence of initial underinvestment is that, by the time that money is spent, there is a partial solution in existence. It may be that the optimal solution now is to throw that away and start again. But it is difficult, if not impossible, to make such a decision. The partial solution becomes a legacy, in the derogatory sense of that word, which means that the eventual solution is constrained to include those bad decisions that led to problems in the first place.

2.3.9 Senior Management

There are many management disaster scenarios ...

At the highest levels of project management, governance is always an issue. There are many projects which find themselves in inappropriate governance structures, and the problems of delivery lower down the organisation are exacerbated by the uninformed decisions made at this level.

To illustrate this phenomenon, consider the scenario that we shall call the parent/child governance scenario, where authority within the project remains with senior management in the customer. Senior management then treats the project team as a parent would a child. They know what they want and, since they are paying, that is what they will have. They ignore the fact that what they want cannot be had at the price they have allocated. This scenario is often seen in large government departments and similar publicly-funded organisations.

Another scenario, also common, we refer to as the enthusiastic builder scenario, where the supplier has a vested interest in prolonging the project, since payment is related to time-on-the-job rather than delivery of results. The supplier will be enthusiatic about agreeing anything with the customer, especially variations in the contract, since that will lead to increased income for the supplier. Where the supplier has a professional responsibility to warn the customer that the requirements they have specified may need to be revisited in future, since they are under-(or over-)specified, they may defer to the customers' authority at the early stage since future renegotiation may lead to further work.

2.3.10 Requirements Explosion

There is a requirements explosion during the lifetime of the project

And of course, because of insufficient planning and simply because projects take a long time from inception to completion, there is a requirements explosion during the lifetime of the project, which means that what is eventually required is significantly different from what was originally anticipated.

This requirements explosion is well known and is attributed to many things, some positive such as realising new potential in the IT that was not anticipated in the initial contract. Such things can be set aside until they are affordable. More pernicious is the slow change in requirements throughout a project that we call requirements drift. We introduce this phenomenon in the next section, where we show that it can be seen as common to all of the reasons for project failure that we have introduced in this section

Requirements Drift

At the outset of a large IT project, even before contracts are agreed, there is a list of requirements stated by the eventual customer. The early stages of the project are all about refining these requirements, at least to the point where teams can be organised and work can begin. A project plan enumerates the requirements and assigns effort and timescales.

But the requirements are poorly understood. Even when the customer signs off a requirements document, it is unlikely that the customer's understanding of what they will get coincides with the supplier's understanding of what they will deliver. It won't be until quite late into the project that a prototype is available and both sides can see the extent of this discrepancy. They will expect it, but they won't be able to define it until that late stage.

As the project proceeds, the organisation, both customer and supplier, learns. One learns about the real potential of the IT, the other about the real critical business processes of the customer. As they learn, they refine the requirements. As they refine them, the requirements drift. Existing requirements change, probably quite subtly. The text of the formal agreement may not change at all, but the understanding of what is meant will shift. New requirements are stated, either because they were overlooked at the outset or because some new aspect of the IT or the customer's business processes is realised. Again, the addition of a single new requirement (when the list of requirements may have thousands of elements) almost goes unnoticed. The requirements have drifted a little further from their original agreement.

Both of these phenomena, revising (our understanding of) existing requirements or adding new ones we refer to as requirements drift, choosing a word which reminds us that the individual changes are almost trivial in the context of the whole.

In the next section we discuss a simple simulation model which (if the simulation can be trusted) shows that a modest drift, where each requirement has a chance of changing of only 0.1% each day can nevertheless lead to a doubling of the cost and the schedule.

Before we look at those models however, let us revisit each of the reasons we have previously given for IT project failure and show how they relate to requirements drift. Then we will argue that, while these reasons will remain, approaching a project knowing that requirements are drifting all the time will help us to mitigate the problems.

Optimism The optimism with which a project begins is based on a (unsafe) conviction that the requirements are better understood than they actually are. As requirements drift slowly, this optimism is not diminished. Each day is not too different from the previous day, so why should we be less optimistic.

Investment Because of our conviction that our grasp on the requirements is solid, we are confident that the initial investment is appropriate. We honestly believe that we can deliver (what we understand of) the requirements for this price. Requirements drift does not impact our belief that this investment is adequate for quite some time.

Procrastination But even when we realise part way into the project (for example, in the roll out of a first prototype) that our optimism was misplaced and the initial investment was inadequate, we renegotiate a new investment (and a new schedule) and make the same mistake again. Now we really believe that we have the project under control. But the requirements continue to drift.

Technical Know-how Lack of technical knowledge exacerbates the problem of believing that we understand the requirements. Indeed, requirements drift may go largely unnoticed simply because the observer isn't sufficiently technically competent to realise the extent to which a drifted requirement differs from an old one.

Human Factors This naive view is actually encouraged by project planning and risk analysis. Neither can be performed competently unless we believe we understand the requirements. Moreover, project planning assumes that the requirements won't change.

Incremental Delivery Of course, well managed projects plan incremental delivery because they understand that it is only when the customer begins to actually evaluate the product that the organisation begins to learn at anything like the necessary rate. Except we don't plan incremental delivery. Because that would be too expensive. Why deliver one component every three months when we can deliver six in a year. And we can do this because we understand the requirements and they are fixed (it says so in the contract). So we go for the big bang. But by the time we do that, the requirements have drifted so far that, even if our product would pass muster against the original requirements, it won't be acceptable because what the customer now wants is not what we thought they wanted at the outset.

Inertia Maybe it is the requirements that are wrong. The initial agreement was to use the IT to support the customer's original business processes. But that is too hard. It would be easier to have the customer adjust their business processes to match what the IT can (and probably does) support. One of the directions of requirements drift is towards this obvious solution. The customer is persuaded that doing things the IT way is a better option (in the sense that they will actually get something) than trying to automate the processes they previously followed.

Legacy When a large IT project goes through a reset, because it has realised that it needs to renegotiate cost and schedule, it inherits a legacy of all the design and building that has gone before. The drift in requirements more-or-less ensures that this legacy is far from an appropriate starting point. So now we have a new problem. How can that legacy be reshaped to supply the new (i.e. the drifted) requirements. It takes courage to be brutal with the legacy. We seldom have that courage. If we were more confident that the requirements drift has been inevitable, then that courage might be easier to find.

Senior Management Good governance in the project is vital to its progress. Where the authority lies determines the direction the project takes. Where senior management believes it understands the requirements, it can compound the problems that arise from all the other factors. Senior management often make political decisions based on a weak understanding of the technology and the requirements. The fact that the requirements are drifting all the while goes even more unnoticed at this level.

Requirements Explosion Because IT seems to offer so much potential, there is always the wish-list of new requirements that would be nice to have. These are the requirements that are properly considered and properly costed. If they are added to the project, the additional resources to deliver them are also added. These are not what we mean by requirements drift. All requirements change subtly over time (even if it is only our late realisation of what was originally intended) so that, while we may end up with only modestly more requirements than we began, most of them will have changed sufficiently that the cost of implementing them greatly exceeds what was originally planned. This is drift. It is drift that we fail to cope with, not explosion.

So, our argment will be that, while we must deal with all the above reasons why large IT projects fail, if we do so always bearing in mind that we have drifting requirements, we will be more cautious about how we proceed and will be more successful. Recognising the existence of requirements drift will mean that a major aspect of the project will be the continual attendance to (our understanding) of the requirements and our constant effort to detect these small changes.

Models that support our argument

To study this phenomenon of requirements drift we have built a series of models using various simulation methods, in particular System Dynamics [Wolstenholme 1990], Analytical Models [Britton 2003] and Monte Carlo methods [Grey 1995]. These models have been of varying degrees of complexity, but all have shown similar behaviour which correlates reasonably with our practical experience [Chatters 1998].

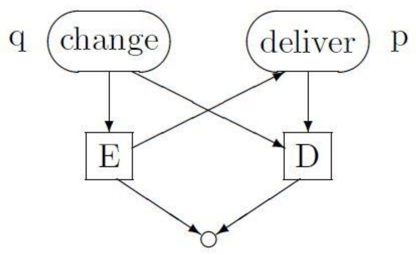

Figure 2: Model of expected (E) and deliv red (D) requirements

The model we will present here has the benefit of being very simple. It shows the basic behaviour that we claim is seen in all large IT projects, but is sufficiently simple that the underlying parameters can be explained without too much mathematical background. The method of simulation is basic Monte Carlo, where the data we present has been averaged over a sufficiently large number of experiments. We have chosen only two sets of parameters to illustrate our point, since that is all we need. All our experiments with this model have shown behaviour that is reasonably represented by these two choices.

The model is shown in Figure 2. We have two significant variables E and D, which respectively refer to the requirements that are expected and delivered. These variables are both lists, indexed by the identity of each requirement. They record whether or not that requirement is expected by the customer and whether or not it has been delivered.

We allow D_i and E_i to change with probability, respectively, p and q.

The variable D is modified on each step with probability p. Only if an individual requirement is expected, will it be delivered and successful delivery will have probability p.The simulation we will illustrate assumes there are 500 of a possible 1000 requirements initially expected. For example if p is 1%, as it is in one of our scenarios below, and if we have around 500 requirements expected, we should deliver about 5 of these with each step, and it should take around 100 steps to deliver all of them (ignoring drift).

We have however built in a very elementary law of diminishing returns [Britton 2003] where the probability of successful delivery is modified by a factor determined by the way requirements to be implemented are chosen. All requirements are chosen, whether they have been implemented or not. This models the fact that changing requirements may involve reimplementation. The net effect, however, is that the number of unimplemented but expected requirements goes down. As it does, so does the expected number that will be successfully implemented at each step.

The changes to E represent revision and understanding of requirements (i.e. drift). A change may remove an old requirement or add a new one. The probability q of this happening to any individual requirement is very low. For example, in the experiments we will describe shortly, it was taken to 0.1%. At each simulation step, each requirement changes with probability q, so that on average (if we have 1000 potential requirements) one requirement will change with each step. Some of those changes will be to respecify (and consequently, unimplement) an expected requirement. Others will be to introduce a new requrement. This represents requirements drift.

time	requirements	Scenario A	Scenario B
	E	D	D
	$q = 0.001$	$p = 0.005$	$p = .010$
0	500	0	0
200	537	321 (60%)	445 (83%)
400	565	453 (80%)	539 (93%)
600	589	515 (87%)	559 (95%)
800	610	550 (90%)	582 (95%)
1000	625	572 (92%)	599 (96%)

Figure 3: Two scenarios, showing requirements drifting over time. Percentages show achievement.

4.1 Experimental Results

The results of a couple of experiments done with this model are shown in Figure 3 and Figure 4. For both scenarios, the rate of drift q was set to 0.1%. For scenario A, the development rate p is 0.5% and for scenario B it is double that, at 1%.

The graph in Figure 4 showns that scenario A, with the lower development rate reaches the initial requirements after around 525 days, but at that time in reality it has only implemented 87% of the drifted requirements. In fact, this development never reaches the drifting requirements. If we accept 95% as our target, it takes considerably more than twice as long to get there.

Scenario B shows the same behaviour. With productivity doubled, it takes nearly 300 days to reach the original requirements and by day 600 we have still only implemented 95% of the drifted requirements.

4.2 How results support our conclusions

The models of course are purely illustrative. They are themselves conjectures. We use them here to show the way that development proceeds and how the modest requirements drift makes an apparently acceptable plan actually return failure according to the criteria that we have established.

In scenario A, for example, the project may well have planned conservatively to deliver at 600 days anticipating the kind of diminishing returns that set in as the product grows. But at 600 days they have ony achieved 87% of what is then understood as the requirements. Anticipating drift, the project would have invested more at the outset and perhaps achieved 95% of requirements by day 600. The amount of additional investment would however be about double, as a comparison between scenarios A and B will support.

It is likely, in practice, of course that realisation that the target is not going to be achieved will be apparent some time before the due date and a correction will be made at that time. This will reduce the total cost somewhat, but it will not be less than if the initial investment had been adequate.

The models are purely indicative. A way of illustrating the kind of behaviour that a large IT project achieves. They are too simple to be true models of real behaviour, although considerably more complex models deliver remarkably similar results.

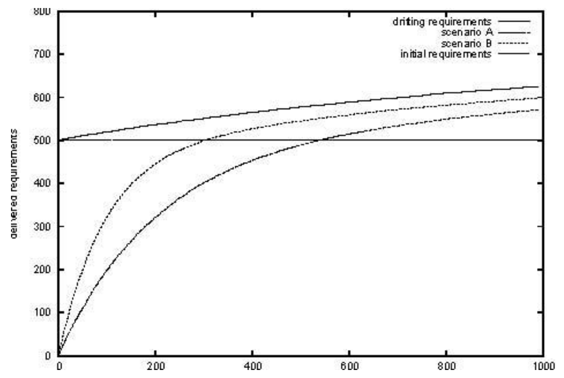

Figure 4: Two scenarios, neither of which achieves what is expected at the outset

5 The way ahead

We claim that a recognition of requirements drift by all parties to the project, but in particular the engineers and their immediate managers will lead to more conservative decisions relating to the plans made for a given investment. Being aware that drift happens and is an inevitable consequence of the need for an organisation to learn, will lead to a project management and execution behaviour that constantly looks for drift. Attending to the requirements, for example by engaging relevant parties in analysis sessions in which the requirements are continuously revisited, will likely expose drift early enough that its effect (of elongating the schedule, increasing the cost or reducing the delivered quality) can at least be anticipated at an earlier stage.

The problem, of course, is believing that drift is so pernicious. It is, after all, just a lot of very small changes, each of which is of little consequence. But if the project team really believes that this is the underlying cause of failure, then they will be prepared to pay more attention to being pedantic about their understanding of requirements. There is no silver bullet here. It's not that simply "involving the customer earlier" will solve the problem. The team have to believe that, even when they do this, the requirements will continue to drift. Indeed, giving too much control to the customer, will simply accelerate the drift.

These observations are just as true when we adopt an Open Systems approach [Krechmer 2005], using components that implement open standards and are to a large extent plug-and-play [Veryard 2000]. This approach may raise the level of complexity with which a project can cope and are normally essential to a modular approach, allowing subsystems developed independently to interoperate. But they do not solve the problem of understanding the requirements and absorbing the drift.

On the other hand, the Open Source movement, which builds on open standards, may have some answers. Here components evolve in the true sense of that word, and the systems they contribute to also evolve. The advantage of this as a development method is that there is no cost associated with production. Schedules still obey some laws of evolution [Henderson 1998] and defy infinite compression, but there is no doubt that quality components do appear from this process. It is difficult to imagine "planning" a system delivery based on the anticipated production of new components by the Open Source method, but it is increasingly common to rely on Open Source components that exist at the outset of a project. The diminishing returns that exacerbate the drift we have observed will not be so apparent in these circumstances, since we will not be paying anything for development.

Conclusions

We have argued that, many of the reasons given for the failure of large IT projects is an inadequte respect for the project's understanding of requirements. We have tried to show that requirements drift either underlies the main reasons given for failure, or contributes to them in a significant way. We have shown, in a model, that the diminishing returns experienced in a large projects as its product gets bigger is exacerbated by drift to the extent that projects may never achieve the kind of quality that makes their product acceptable. And finally, while offering no silver bullet, we have suggested that a true belief in requirements drift among engineers and their managers may itself go a long way towards mitigating its effects.

References

[ACM] ACM Committee on Computers and Public Policy, The Risks Forum. http://catless.ncl.ac.uk/risks

[Axelrod 99] Axelrod, R and M. D. Cohen, Harnessing Complexity -Organisational Implications of a Scientific Frontier. Free Press, NY, 1999

[BCS 2004] British Computer Society, The Challenges of Complex IT Projects. The report of a working group from the Royal Academy of Engineering and the British Computer Society, British Computer Society, 2004

[Boehm 1981] Boehm, B, Software engineering economics.. Prentice-Hall, Englewood Cliffs, NJ, 1981

[Boehm 2000] Boehm, B et al, Software cost estimation with Cocomo II. Prentice-Hall, Englewood Cliffs, NJ, 2000

[Britton 2003] Britton, N. F, Essential Mathematical Biology. Springer-Verlag, London, 2003

[Brookes 1979] Brookes, F. P, The Mythical Man-month. Addison Wesley, 1979

[Cambridge 2001] Cambridge Reporter, CAPSA and its implementation. Cambridge Reporter, 2/11/01, http://www.admin.cam.ac.uk/reporter

[Chapman 2002] Chapman, J, System Failure. DEMOS, 2002

[Chatters 1998] Chatters, B , P. Henderson and C. Rostron, SIMMER: Software and Systems Integration Modelling Metrics and Risks (Getting to Level 4). EuroSPI 98, Goteborg, November 1998

[Grey 1995] Grey, S, Practical Risk Assessment for Project Management. Wiley, 1995

[Hayes 2003] Hayes, B, On the Threshold. American Scientist, Vol.91, Jan 2003

[Henderson 1998] Henderson, P, Laws for Dynamic Systems. Proceedings of International Conference on Software Reuse (ICSR 98), IEEE Computer Society Press, 1998

[Holyrood 2004] Holyrood Inquiry, The Holyrood Inquiry. A Report by The Rt Hon Lord Fraser of Carmyllie QC, http://www.holyroodinquiry.org, 2004

[Humphrey 2004] Humphrey, W.S, Why Big Software Projects Fail: The 12 Key Questions.The Journal of Defense Software Engineering, 2004

[Intellect 2003] Intellect, IT Supplier -Code of Best Practice. http://www.intellectuk.org, 2003

[Kauffman 2000] Kauffman, S, Investigations. OUP, 2000

[Krechmer 2005] Krechmer, K, The Meaning of Open Standards. The International Journal of IT Standards and Standardization Research, Vol. 4 No. 1, 2005.

[Neumann 1995] Neumann, P, Computer Related Risks. Addison Wesley, 1995

[NAO 2006] National Audit Office, National Programme for IT in the NHS Report by the Comptroller and Auditor General, HC 1173, http://www.nao.org.uk, 2006

[NHS 2005] National Health Service, National Programme for IT in the NHS http://www.connectingforhealth.nhs.uk/, 2005

[Petroski 1992] Petroski, H, To Engineer is Human -The role of failure in successful design. Vintage Books, 1992

[Sauer 2003] Sauer, C and C. Cuthbertson, The State of IT Project Management in the UK 20022003. Templeton College, Oxford, www.cw360ms.com/pmsurveyresults/surveyresults.pdf, 2003

[Veryard 2000] Veryard, R, The Component Based Business -Plug and Play. Springer Practitioner Series

[Wolstenholme 1990] Wolstenholme, E, System Enquiry, A System Dynamic Approach.John Wiley & Sons, New York 1990

[Xia 2004] Xia, W and G. Lee Grasping the Complexity of IS Development Projects. Comm. ACM, Vol.47, No.5, 2004