

## Software Project Risks

Gillian Adens, Robert Armstrong

Tassc Limited  
[www.tassc-solutions.com](http://www.tassc-solutions.com)

First Published: August 2001  
Last Updated: January 2008

Copyright 2001-2009, Tassc Limited. All Rights Reserved.

*There are many potential risks and pitfalls facing IT projects. So many projects fail for one reason or another – they deliver too late, run out of budget or just don't meet basic business requirements. One of the primary reasons for so many project failures is that all too few software projects undertake any risk assessment. Therefore no account is taken of how risks might impact the project.*

*In this paper we draw from years of consulting experience to catalogue the most common risks in today's software development projects. By being aware of these risks it is possible to actively manage a project to negate, or at least mitigate, the impact of risks. We can also develop contingency plans so that we can take immediate action should any of the risks fire.*

*Software project management is all about making the best possible plans based on the information available at the time, and then proactively tracking and controlling projects in response to deviation from plan. Risk assessment and contingency planning is a major component of this.*

*Typically we have found that software project risks fall into one of four categories – risks associated with people, project scope, software development processes or tools and technologies.*



### The Right People

---

Software development is still a labour intensive activity. The quantity and quality of software developed often directly relates to the skills and availability of development staff. Resourcing a software project can be fraught with potential risks. IT staff can be expensive and up-to-date skills are often in high demand.

*supplement missing skills with training, consulting and mentoring*

One risk is simply not having enough of the right sort of resources available to a project at critical times - this can severely delay the project or reduce the quality of the delivered system. For example, it is essential to have an experienced software engineer available early in the project to oversee key technology decisions and to define the software architecture to provide a framework for quality design. Similarly, it is sensible not to leave all the testing to the end of the project.

*choose project team members to ensure a useful mix of skills*

A good mix of skills is required. Even the most skilled Java programmers will not necessarily develop a robust and maintainable mission critical system without the involvement of experienced analysts and designers.

Every project team needs a natural leader to provide the team with the necessary drive and commitment to deliver on schedule. However, there is a risk of having too many 'experts' or too many drivers in the one team. The best teams are formed from a mix of skills - too many leaders can result in differences of opinion, confrontation and confusion, actually slowing down progress.

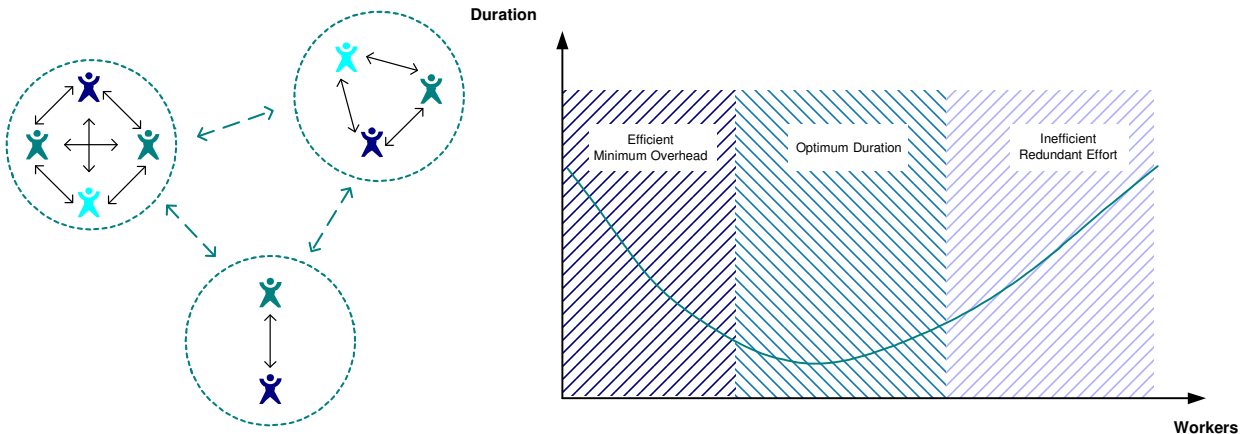
*calculate the optimum number of resources given the project size*

As well as the risk of not having enough developers there is an equal, and sometimes more damaging risk, of over-resourcing a project. One mistake often made is simply to throw developers at a late project in the hope that 'many hands will make light work'. In fact new people can slow down a project team as they take time to come up-to-speed with the requirements, design and structure of the code.

*encourage joint working to spread knowledge*

Retaining key staff can be critical to the success of a project. Financial incentives are rarely enough. Technical staff are often equally motivated by new and challenging technology, and the opportunity to see their software providing a useful business solution. It is important to ensure that skills and knowledge are shared so that no individual becomes indispensable.

## Working Together



*plan the dependencies between teams to allow parallel working*

Modern IT projects are rarely small enough to be undertaken by individuals. Teams and whole departments can be involved in a single project. It is high risk to have large numbers of developers working on a project without a clear separation of responsibilities.

It is advantageous to partition development effort and have separate teams work on different aspects of a project. Good organisation of people can reduce duplication of effort and ensure that all staff are working effectively and efficiently.

Another risk is leaving integration until the last moment, only to discover that separate teams have been working with incompatible designs or technologies.

*schedule for a level of communication and management overhead*

There are many risks relating to communications in a team environment. As more people are added to the project the lines of communication and management overheads naturally increase.

*co-locate team members and provide isolation from distractions*

There are risks associated with having communication and reporting mechanisms that are too informal or at the other end of the scale overly formal - a balance has to be struck. If things are too informal it can lead to confusion and a lack of quality. If things are too formal it can kill spontaneity and creativity, and de-motivate staff.

*plan effective communication mechanisms*

Focus is also important. Teamwork is essential and if the team is fragmented or staff are pulled in too many directions, it is likely that the project will suffer. Likewise location and environment is an important factor. The team needs isolation from disruption and facilities to allow them to work effectively together.

The people factor is so important that even personality clashes can seriously impact the effectiveness of a team. So can other soft factors such as language and culture. Many projects these days span the globe with collaborative developments between separate companies perhaps even located in separate countries. Different time zones and languages make communication more difficult. It becomes impossible to organise a quick meeting to resolve a design issue. Even where the language is common, cultures can differ, and lead to misunderstanding.



## Knowing What to Develop

---

If you don't know what you are trying to build it is impossible to build the correct solution. All too often excellent engineers build great software that simply fails to meet business needs. One of the greatest risks at the outset of any new project is not having a clear enough idea of what is required of the final system. This leads to a serious risk of under-estimating how much work there is to do.

*build functional prototypes to clarify details and usability issues*

Generally the risk is in not understanding the business functionality in enough detail. However sometimes the risks relate to not understanding the business context within which the new system will operate. This can lead to a lack of appreciation of technical issues such as what platform the system needs to be delivered on and the business need for basic levels of system performance. We have all seen systems that deliver perfect functionality but are unusable because they simply run too slowly on the customer's hardware.

*bid on the initial scoping work, not on the whole project*

A great deal of software development these days is out-sourced. Many companies are bidding for work on very sketchy requirements statements or requests for tender. In this situation there is a clear risk that no-one understands the scope of the work - not even the client. This can ultimately lead to financial penalties or even litigation. These are the situations that are increasingly hitting the headlines of our trade press.

*agree required levels of user involvement at the start of the project*

Another major risk is not having access to the people who hold the business knowledge - typically the end users. Often these people will be part of a different department or even a different company and will almost certainly work at a separate location. They have their own work demands and pressures. Therefore time available to the software project can be extremely limited - causing hold-ups in the decision-making process.

*JAD sessions can involve end users from all aspects of the business*

Sometimes business knowledge is provided in the form of a project sponsor. This can be positive as it saves having to canvass the opinions of a large group of users and the project sponsor often feels a higher level of commitment to the project. The risk is that not all the stakeholders are sufficiently involved. The project sponsor may understand some of the business requirements but is unlikely to have the breadth of knowledge to cover all aspects of a large system.

## With Clear Requirements

---



*use case modeling can ensure requirements are stated at an appropriate level of detail*

The scenario above clearly illustrates the risks relating to scope definition. This is a typical situation where the engineer's perception of the project does not match that of the customer. Such misunderstanding is very common in software projects and will inevitably lead to tension later in the project as project timescales and costs have to be substantially adjusted. It may even result in the project being cancelled. Poorly defined scope at the outset of a project is one of the most common reasons for failure.

However, it is almost impossible to accurately specify a system with sufficient detail and clarity at the very start of the project. What is needed is a formal method of describing requirements so that the agreed project scope is unambiguous to all parties. Then everyone has an agreed basis on which to progress and project plans and schedules can be created with confidence. As the project unfolds and further requirements analysis reveals additional scope, there is sufficient justification for changing schedules and plans to take account of the additional development effort required.

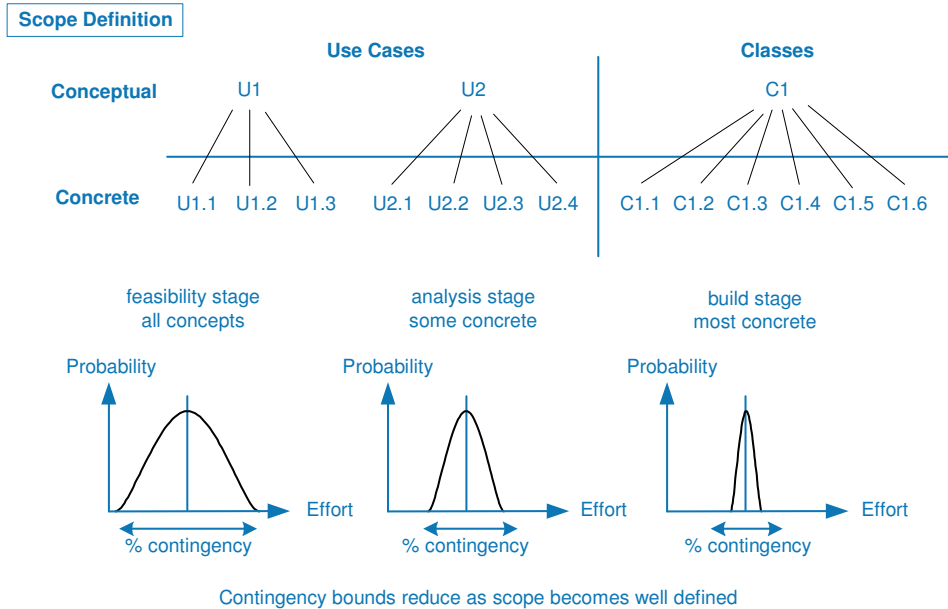
## Analysing and Discovering Detail

---

*build in contingency to early project plans and schedules*

No matter what the eventual solution, the process of refining scope understanding from a conceptual analysis view to concrete artifacts is a journey of discovery. The population of software artifacts tends to increase. The risk is not understanding this and failing to schedule an appropriate level of effort or build in an appropriate level of contingency.

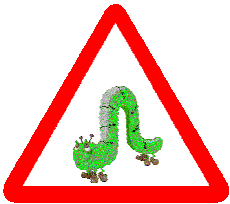
At the outset the project is often described in very general terms as a statement of high level business requirements - for example the system may have to hold customer details. During analysis it is discovered that there are various types of customers to be modelled - for example different details are held for retail and business customers. As the detailed design is produced these analysis level classes are translated into concrete classes for implementation. The number of resulting implementation classes is generally a function of the implementation technology, software architecture and deployment strategy. For example there may be user interface classes, business classes and persistent classes all supporting the concept of retail customer information.



*use appropriate metrics to estimate at each stage*

At the early analysis stage of the project the effort metrics used should be larger and the contingency more significant. At the detailed design stage the effort metrics can be smaller and the contingency can be reduced. The further into a project, the more definite and confident you will be about the effort involved.

An incremental development process allows the team to discover and understand one aspect of scope at a time, and project managers to gather intrinsic evidence on appropriate effort metrics to use for scheduling purposes.



### Controlling Scope Creep

It's one thing to understand the project requirements but will they remain stable?

Change is almost inevitable. Business needs are continually evolving and we live in a highly competitive society.

*catalogue any change requests and measure the impact of each change*

Most development processes these days actively encourage user feedback. This is very positive as the involvement of end users helps to shape the application and ensure that useful business functionality is delivered.

One risk is that the priority of business requirements continually changes. This can be as a result of changes to the business direction or re-evaluation of the market requirements.

The other risk is that new requirements are introduced. This can be as a result of involving new users in project reviews, or in response to competition or new legislation.

*actively manage change and schedule according to priority*

The challenge is to control this process and ensure that it does not swamp the development team. It is impossible to design and implement a system when requirements are endlessly changing, particularly if the changes are major.

The risk associated with changing requirements is that the development team loses focus, or that the changes are so major that they invalidate the design so far. This can have a significant impact on timescales and costs, and a demoralising effect on the development team. One way of handling this effectively is to allow the development team to continue work on the current iteration without interruption and to allocate change requests to future iterations according to their priority.



## Within a Well-Defined Development Process

---

An appropriate and well-defined process is an essential ingredient for a successful software project.

One risk is simply not having any process in place at all. Software projects these days are just too complex and too business-critical to allow a group of developers to 'hack out a solution'. Without a documented and communicated process and related standards, quality of the delivered software will be compromised.

*match the process to the type of development activity*

However, it's not just a case of *any* process. The chosen process has to be fit for purpose. There is a risk of choosing the wrong process.

Trying to use an overly formal process for a quick GUI prototype is counterproductive. The process becomes cumbersome and slows down what should be a highly dynamic and creative activity. Many web development companies find that RUP falls into this category. They tend to prefer processes such as RAD or Extreme Programming.

On the other hand more informal processes such as RAD might not be appropriate for developing the technical infrastructure for a complex client-server system. DSDM has been found to be an excellent approach for commercial systems that are focused on front-end GUI functionality. However a real-time business critical application would not be the ideal candidate for this type of development process. RUP might be a better choice in this situation.

*document responsibilities and feedback mechanisms*

Another risk in software development is failure to allow or react quickly to feedback. No project will progress perfectly and without issue. An overly cumbersome feedback mechanism is detrimental to software development.

Quality is never automatic. Compliance to standards has to be encouraged and verified through design reviews and testing. Quality will suffer if the process does not define responsibilities and points of quality control.



## Using Appropriate Tools and Technologies

---

Tools and technologies need to be fit for purpose. Certain programming languages are more appropriate for certain types of development. For example, Visual Basic is very suitable for GUI developments but less so for complex real-time calculations.

*choose 'tried and tested' tools and technologies*

Immature technologies are clearly high risk. Sometimes we need to take such risks in order to innovate. However it is preferable to use tried and tested technologies where possible. Early adopters spend much of their time ironing out the bugs and discovering the limitations of new technologies.

*examine vendor support and stability*

Risks associated with newer technologies include stability and levels of support and maintenance. The stability and longevity of the vendor could also represent a risk to the project.

*build technical prototypes*

There may be some specific technical or performance issues that remain untested. It is a high-risk strategy to leave these until late in the project, particularly if they would represent a project 'show stopper'.

Poor configuration management is a high risk. It is important to be able to backtrack at any point in the development life-cycle to some previous stable state.

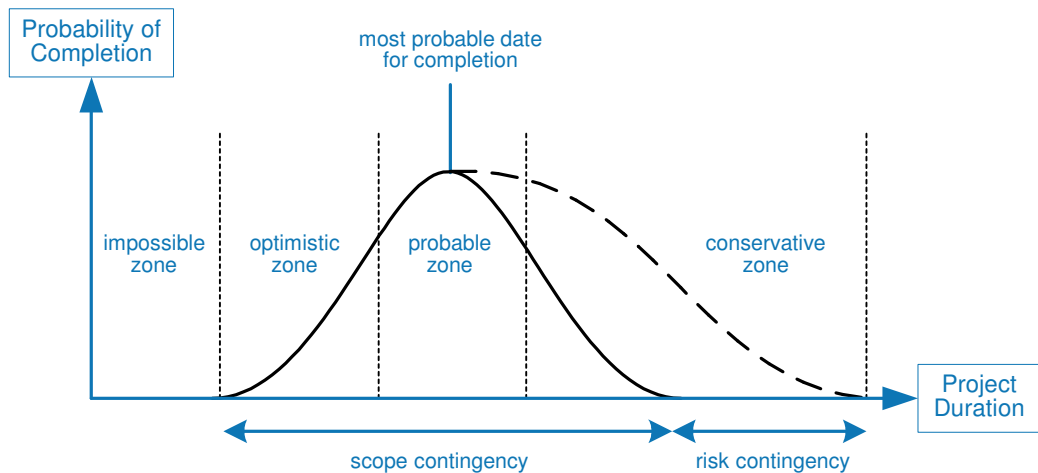
*Carefully plan software releases to fit with business operations*

There are risks associated with the release of software and implementation within the business. Software is better received if it is perceived as providing an advantage over existing systems. It is important that new users have sufficient time to dedicate to learning how to use the new software and that they receive proper support and training. Bugs discovered after release need to be addressed in a timely fashion or users can become disillusioned.

## Assessing Project Risk

So what do we do about all of these potential risks? Clearly we should identify and analyse the risks. By considering the impact of each risk as well as its likelihood of firing, it is possible to calculate the level of risk exposure.

The more risk associated with a project the more likely it is for the project to take longer to complete. The total risk exposure can be added to initial estimates to calculate a conservative delivery date for the project. By analysing the average probability and impact of all the risks a contingency can be calculated to estimate the most likely project duration.



It is therefore possible to plot the duration of a project against the probability of completion. The resulting graph can be segmented into 4 areas:

The 'impossible' zone – given the amount of scope, the project simply cannot be completed this quickly.

The 'optimistic' zone – the earliest possible delivery dates assuming the project runs smoothly and that scope is well defined and stable.

The 'probable' zone – the most likely delivery dates for the project.

The 'conservative' zone – pessimistically the project may take this long if scope creep is experienced and other risks fire.