Part I

Introduction

CHAPTER 1

Evaluating Risk

Here, I am going to re-cap on some pre-existing risk management theory in order to set the scene for the next chapter which heads back to looking at risk on software projects.

1.1 Risk Registers

Most developers are familiar with recording issues in an issue tracker. For all of the same reasons, it's good practice to record the risks you face running a project or an operation in a Risk Register[1]. Typically, this will include for each risk:

- The **name** of the risk, or other identifier.
- A **categories** to which the risk belongs (this is the focus of the Risk Landscape chapter in Part 2).
- A **brief description** or name of the risk to make the risk easy to discuss
- Some estimate for the **Impact**, **Probability** or **Risk Score** of the risk.
- Proposed actions and a log of the progress made to manage the risk.

Some points about this description:

A Continuum of Formality

Remember back to the Dinner Party example at the start: the Risk Register happened *entirely in your head*. There is a continuum all the way from "in your head" through "using a spreadsheet" to dedicated Risk Management software.

[1] https://en.wikipedia.org/wiki/Risk_register

It's also going to be useful *in conversation*, and this is where the value of the Risk-First approach is: providing a vocabulary to *talk about risks* with your team.

Probability And Impact

Probability is how likely something is to happen, whilst **Impact** is the cost (usually financial) when it does happen.

In a financial context (or a gambling one), we can consider the overall **Risk Score** as being the sum of the **Impact** of each outcome multiplied by it's **Probability**. For example, if you buy a 1-Euro ticket in a raffle, there are two outcomes: win or lose. The impact of *winning* would be (say) a hundred Euros, but the **probability** might be 1 in 200. The impact of *losing* would be the loss of 1 Euro, with

Outcome	Impact	Probabilty	Risk Score
Win	+ 99 EUR	1 in 200	.5 EUR
Lose	- 1 EUR	199 in 200	-.99 EUR

Risk Management in the finance industry *starts* here, and gets more complex, but often (especially on a software project), it's better to skip all this, and just figure out a Risk Score. This is because if you think about "impact", it implies a definite, discrete event occurring, or not occurring, and asks you then to consider the probability of that occurring.

Risk-First takes a view that risks are a continuous quantity, more like *money* or *water*: by taking an action before delivering a project you might add a degree of Schedule Risk, but decrease the Operational Risk later on by a greater amount.

1.2 Risk Matrix

A risk matrix presents a graphical view on where risks exist. Here is an example, showing the risks from the dinner party in the A Simple Scenario chapter:

This type of graphic is *helpful* in deciding what to do next, although alternatively, you can graph the overall **Risk Score** against the Pay-Off. Easily mitigated risk (on the right), and worse risks (at the top) can therefore be dealt with first (hopefully).

Figure 1.1: Risk Register of Dinner Party Risks

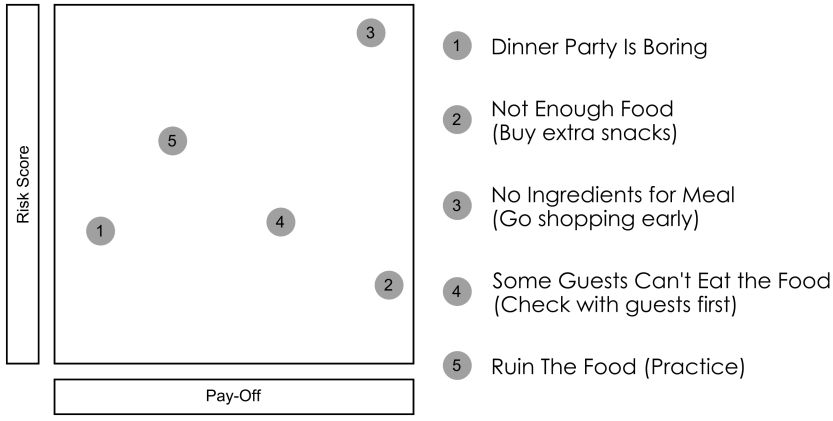

Figure 1.2: Risk Register of Dinner Party Risks, Considering Pay-Off

1.3 Unknown Unknowns

One of the criticisms of the Risk Register approach is that of mistaking the map for the territory. That is, mistakenly believing that what's on the Risk Register *is all there is*.

In the preceding discussions, I have been careful to point out the existence of Hidden Risks for that very reason. Or, to put another way:

> What we don't know is what usually gets us killed - Petyr Baelish, *Game of Thrones*

Donald Rumsfeld's famous Known Knowns[2] is also a helpful conceptualisation:

- **A *known* unknown** is an Attendant Risk. i.e. something you are aware of, but where the precise degree of threat can't be established.
- **An *unknown* unknown** is a Hidden Risk. i.e a risk you haven't even thought to exist yet.

1.4 Risk And Uncertainty

Arguably, this site uses the term 'Risk' wrongly: most literature suggests risk can be measured[3] whereas uncertainty represents things that cannot.

I am using **risk** everywhere because later we will talk about specific risks (e.g. Boundary Risk or Complexity Risk), and it doesn't feel grammatically correct to talk about those as **uncertainties**, especially given the pre-existing usage in Banking of terms like Operational Risk[4] or Reputational risk[5] which are also not really a-priori measurable.

1.5 The Opposite Of Risk Management

Let's look at the classic description of Risk Management:

> Risk Management is the process of thinking out corrective actions before a problem occurs, while it's still an abstraction. The opposite of risk management is crisis management, trying to figure

[2] https://en.wikipedia.org/wiki/There_are_known_knowns

[3] https://keydifferences.com/difference-between-risk-and-uncertainty.html

[4] https://en.wikipedia.org/wiki/Operational_risk

[5] https://www.investopedia.com/terms/r/reputational-risk.asp

> out what to do about the problem after it happens. - Waltzing With Bears, *Tom De Marco & Tim Lister*[6]

This is not how Risk-First sees it:

First, we have the notion that Risks are discrete events, again. Some risks *are* (like gambling on a horse race), but most *aren't*. In the Dinner Party, for example, bad preparation is going to mean a *worse* time for everyone, but how good a time you're having is a spectrum, it doesn't divide neatly into just "good" or "bad".

Second, the opposite of "Risk Management" (or trying to minimise the "Down-side") is either "Upside Risk Management", (trying to maximise the good things happening), or it's trying to make as many bad things happen as possible.

Third, Crisis Management is *still just Risk Management*: the crisis (Earthquake, whatever) has *happened*. You can't manage it because it's in the past. All you can do is Risk Manage the future (minimize further casualties and human suffering, for example).

Yes, it's fine to say "we're in crisis", but to assume there is a different strategy for dealing with it is a mistake: this is the Fallacy of Sunk Costs[7].

1.6 Invariances #1: Panic Invariance

You would expect then, that any methods for managing software delivery should be *invariant* to the level of crisis in the project. If, for example, a project proceeds using Scrum[8] for eight months, and then the deadline looms and everyone agrees to throw Scrum out of the window and start hacking, then *this implies there is a problem with Scrum*, and that it is not *Panic Invariant*. In fact, many tools like Scrum don't consider this:

- If there is a production outage during the working week, we don't wait for the next Scrum Sprint to fix it.
- Although a 40-hour work-week *is a great idea*, this goes out of the window if the databases all crash on a Saturday morning.

In these cases, we (hopefully calmly) *evaluate the risks and Take Action*.

This is **Panic Invariance**: your methodology shouldn't need to change given the amount of pressure or importance on the table.

[6] http://amzn.eu/d/i0IDFA2

[7] https://en.wikipedia.org/wiki/Escalation_of_commitment

[8] https://en.wikipedia.org/wiki/Scrum_(software_development)

1.7 Invariances #2: Scale Invariance

Another test of a methodology is that it shouldn't fall down when applied at different *scales*. Because, if it does, this implies that there is something wrong with the methodology. The same is true of physical laws: if they don't apply under all circumstances, then that implies something is wrong. For example, Newton's Laws of Motion fail to calculate the orbital period of Mercury, which led to Einstein trying to improve on them with the Theory of Relativity[9].

Some methodologies are designed for certain scales: Extreme Programming[10] is designed for small, co-located teams. And, that's useful. But the fact it doesn't scale tells us something about it: chiefly, that it considers certain *kinds* of risk, while ignoring others. At small scales, that works ok, but at larger scales, other risks (such as team Coordination Risk) increase too fast for it to work.

So ideally, a methodology should be applicable at *any* scale:

- A single class or function
- A collection of functions, or a library
- A project team
- A department
- An entire organisation

If the methodology *fails at a particular scale*, this tells you something about the risks that the methodology isn't addressing. It's fine to have methodologies that work at different scales, and on different problems. One of the things Risk-First explores is trying to place methodologies and practices within a framework to say *when* they are applicable.

1.8 Value vs Speed

"Upside Risk"

"Upside Risk" isn't a commonly used term: industry tends to prefer "value", as in "Is this a value-add project?". There is plenty of theory surrounding **Value**, such as Porter's Value Chain[11] and Net Present Value[12]. This is all fine so long as we remember:

[9] https://en.wikipedia.org/wiki/Theory_of_relativity

[10] https://en.wikipedia.org/wiki/Extreme_programming

[11] https://en.wikipedia.org/wiki/Value_chain

[12] https://en.wikipedia.org/wiki/Net_present_value

- **The probability of Pay-Off is risky**: Since the value is created in the future, we can't be certain about it happening - we should never consider it a done-deal. **Future Value** is always at risk. In finance, for example, we account for this in our future cash-flows by discounting them according to the risk of default.
- **The Pay-Off amount is risky**: Additionally, whereas in a financial transaction (like a loan, say), we might know the size of a future payment, in IT projects we can rarely be sure that they will deliver a certain return. On some fixed-contract projects this sometimes is not true: there may be a date when the payment-for-delivery gets made, but mostly we'll be expecting an uncertain pay-off.
- Humans tend to be optimists (especially when there are lots of Hidden Risks), hence our focus on Downside Risk. Sometimes though, it's good to stand back and look at a scenario and think: am I capturing all the Upside Risk here?

Speed

For example, in Rapid Development[13] by Steve McConnell we have the following diagram:

And, this is *fine*, McConnell is structuring the process from the perspective of *delivering as quickly as possible*. However, here, I want to turn this on it's head. Software Development from a risk-first perspective is an under-explored technique, and I believe it offers some useful insights. So the aim here is to present the case for viewing software development like this:

As we will see, *Speed* (or Schedule Risk as we will term it) is one risk amongst others that need to be considered from a risk-management perspective. There's no point in prioritising *speed* if the software fails in production due to Operational Risk issues and damages trust in the product.

Eisenhower's Box

Eisenhower's Box is a simple model allowing us to consider *two* aspects of risk at the same time:

- How valuable the work is (Importance, Value).
- How soon it is needed (Urgency, Time).

The problem is, we now need to take a call on whether to do something that is *urgent* or something that is *important*.

[13] `http://a.co/d/ddWGTB2`

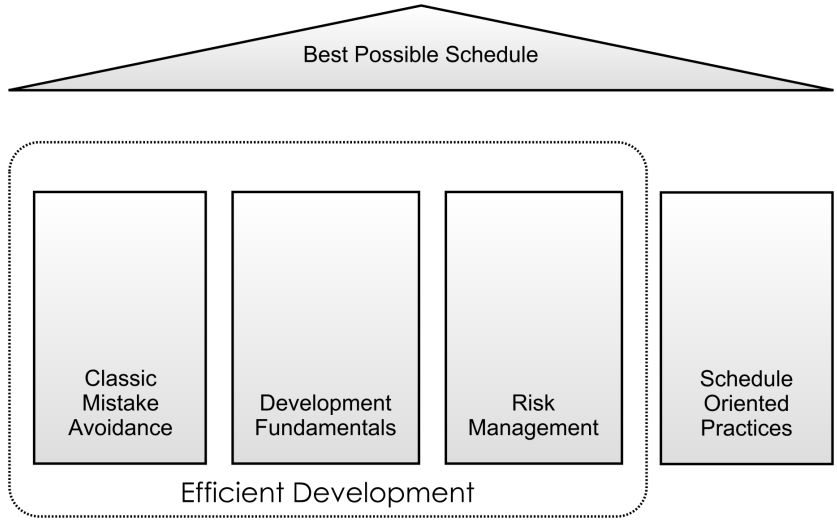

Figure 1.3: Pillars, From Rapid Development By Steve McConnell

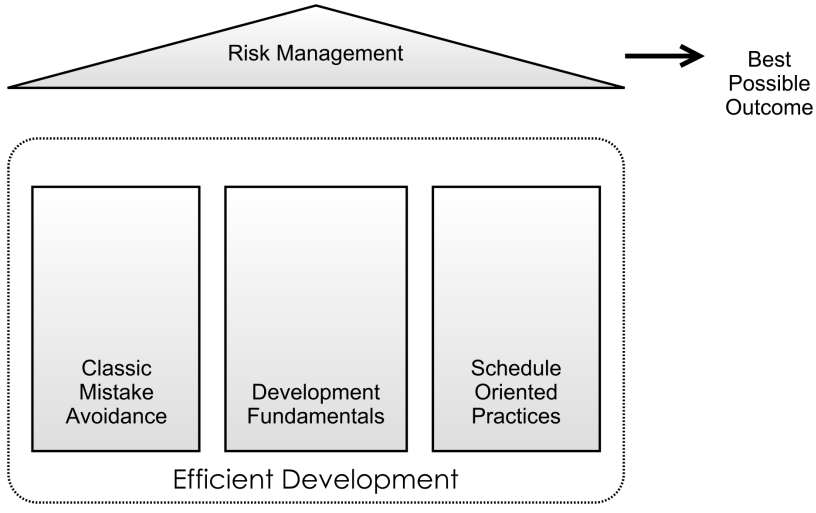

Figure 1.4: Pillars, re-arranged

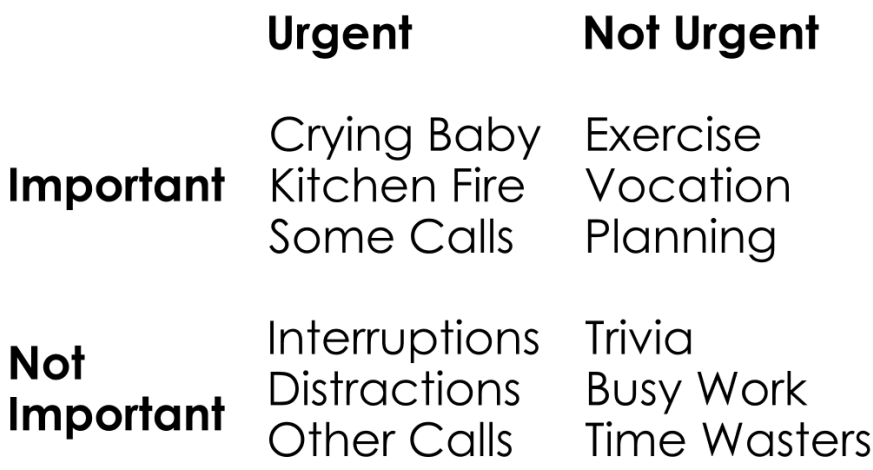

***Figure 1.5: A basic "Eisenhower box" to help evaluate urgency and importance. Items may be placed at more precise points within each quadrant. - Adapted From Time Management,* Wikipedia[14]**

1.9 Discounting

Net Present Value allows us to discount value in the future, which offers us a way to reconcile these two variables. The further in the future the value is realised, the bigger the discount. This is done because payment *now* is better than payment in the future: there is the risk that something will happen to prevent that future payment. This is why we have *interest rates* on loan payments.

In the diagram, you can see two future payments, Payment **A** of £100 due in one year, and Payment **B** of £150 due in 10 years. By discounting at a given rate (here at a high rate of 20% per year) we can compare their worth *now*. At this discount rate, Payment **A**, - arriving next year - has a far greater value.

Can we do the same thing with risk? Let's introduce the concept of Net Present Risk, or NPR:

> Net Present Risk is the *Impact* of a Future risk, discounted to a common level of *Urgency*.

Let's look at a quick example to see how this could work out. Let's say you had the following risks:

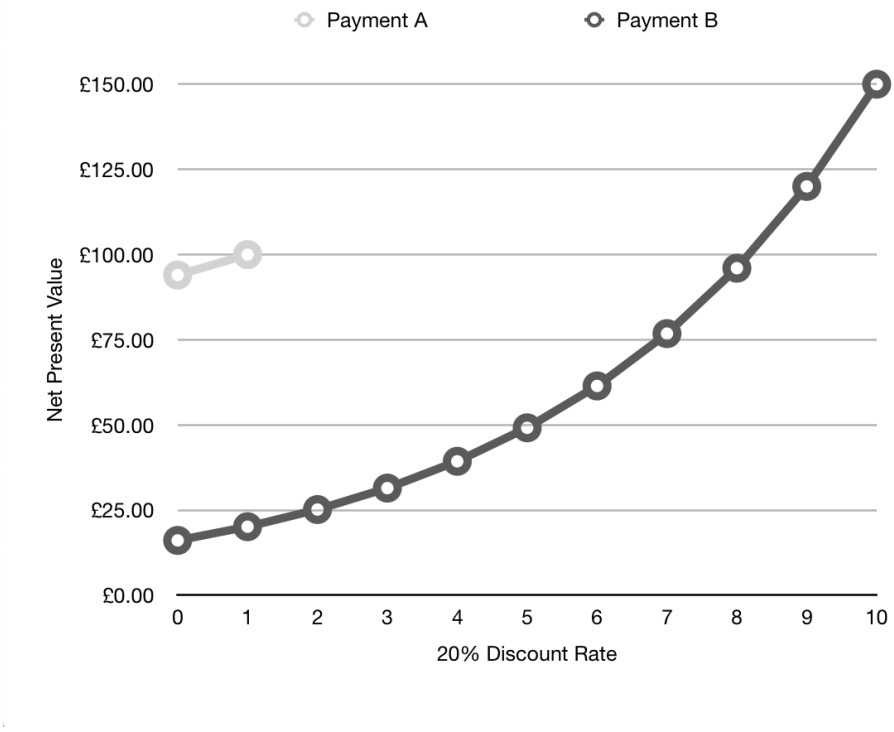

Figure 1.6: Net Present Value Discounting

- Risk **A**, which will cost you £50 in 5 day's time.
- Risk **B**, which will cost you £70 in 8 day's time.

Which has the biggest NPR? Well, it depends on the discount rate that you apply. Let's assume we are discounting at 6% per *day*. A graph of the discounted risks looks like this:

On this basis, the biggest NPR is **B**, at about £45. If we increase the discount factor to 20%, we get a different result:

Now, risk **A** is bigger.

Because this is *Net* Present Risk, we can also use it to make decisions about whether or not to mitigate risks. Let's assume the cost of mitigating any risk *right now* is £40. Under the 6% regime, only Risk **B** is worth mitigating today, because you spend £40 today to get rid of £45 of risk (today).

Under the 20% regime, neither are worth mitigating. The 20% Discount Rate may reflect that sometimes, future risks just don't materialise.

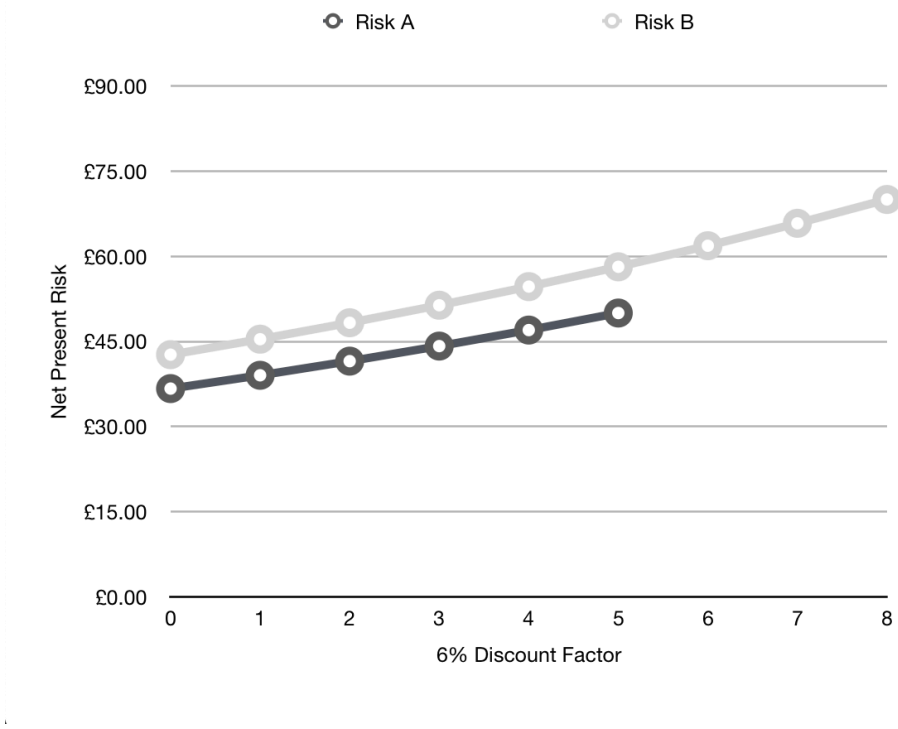

Figure 1.7: Net Present Risk, 6% Discount Rate

Discounting the Future To Zero

I have worked in teams sometimes where the blinkers go down, and the only thing that matters is *now*. Anything with a horizon over a week is irrelevant. Regimes of such hyper-inflation[15] are a sure sign that something has *really broken down* within a project. Consider in this case a Discount Factor of 60% per day, and the following risks:

- Risk A: £10 cost, happening *tomorrow*
- Risk B: £70 cost, happening in *5 days*.

Risk B is almost irrelevant under this regime, as this graph shows:

Why do things like this happen? Often, the people involved are under incredible job-stress: usually they are threatened on a daily basis, and therefore feel they have to react. In a similar way, publicly-listed companies also often

[15] https://en.wikipedia.org/wiki/Hyperinflation

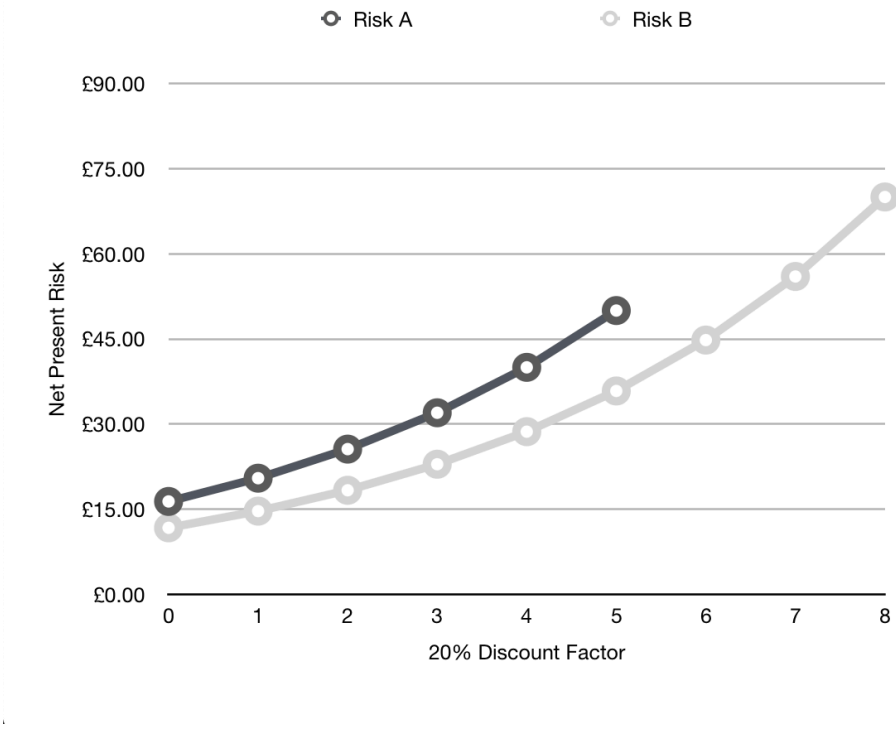

Figure 1.8: Net Present Risk, 20% Discount Rate

apply short-term focus, because they only care about the *next annual report*, which limits their horizons and ability to consider future risk.

Under these circumstances, we often see *Pooh-Bear Procrastination*:

> "Here is Edward Bear coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way. . . if only he could stop bumping for a moment and think of it!"
>
> —A. A. Milne, *Winne-the-Pooh*[16]

1.10 Is This Scientific?

Enough with the numbers and the theory: Risk-First is an attempt to provide a practical framework, rather than a scientifically rigorous analysis. For

[16] http://amzn.eu/d/acJ5a2j

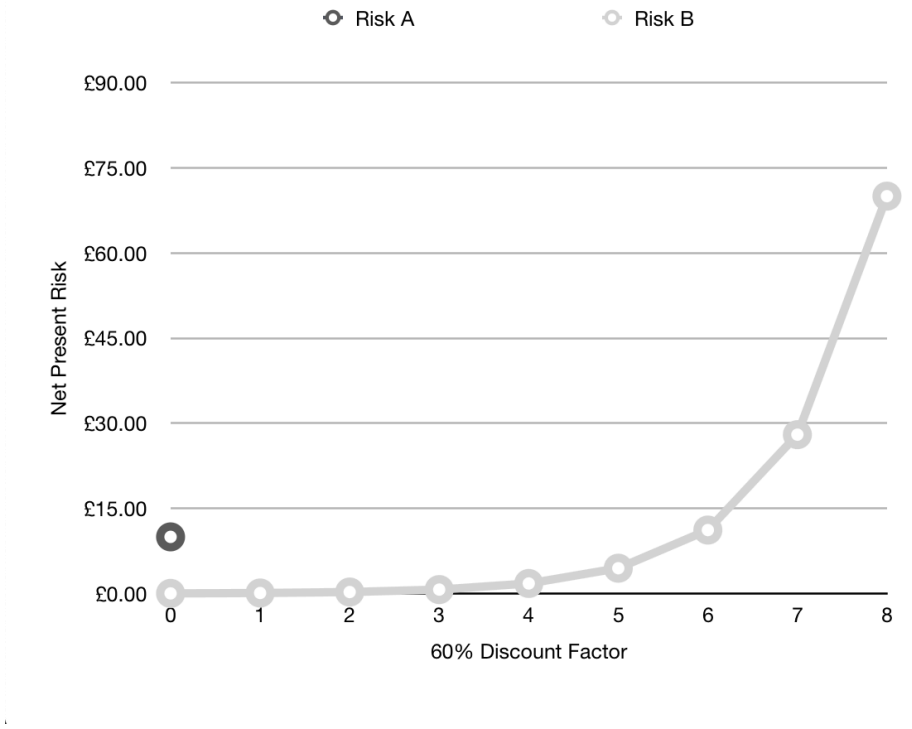

Figure 1.9: Net Present Risk, 60% Discount Rate

software development, you should probably *give up* on trying to compute risk numerically. You *can't* work out how long a software project will take based purely on an analysis of (say) *function points*. (Whatever you define them to be).

- First, there isn't enough scientific evidence for an approach like this. We *can* look at collected data about IT projects, but techniques and tools advance rapidly.
- Second, IT projects have too many confounding factors, such as experience of the teams, technologies used etc. That is, the risks faced by IT projects are *too diverse* and *hard to quantify* to allow for meaningful comparison from one to the next.
- Third, as soon as you *publish a date* it changes the expectations of the project (see Student Syndrome).
- Fourth, metrics get misused and gamed (as we will see in a later chapter).

Reality is messy. Dressing it up with numbers doesn't change that and you

risk fooling yourself. If this is the case, is there any hope at all in what we're doing? Yes: *forget precision*. You should, with experience be able to hold up two separate risks and answer the question, "is this one bigger than this one?"

With that in mind, let's look at how we can meet reality as fast and often as possible.

Part II

Risk

Part III

Application