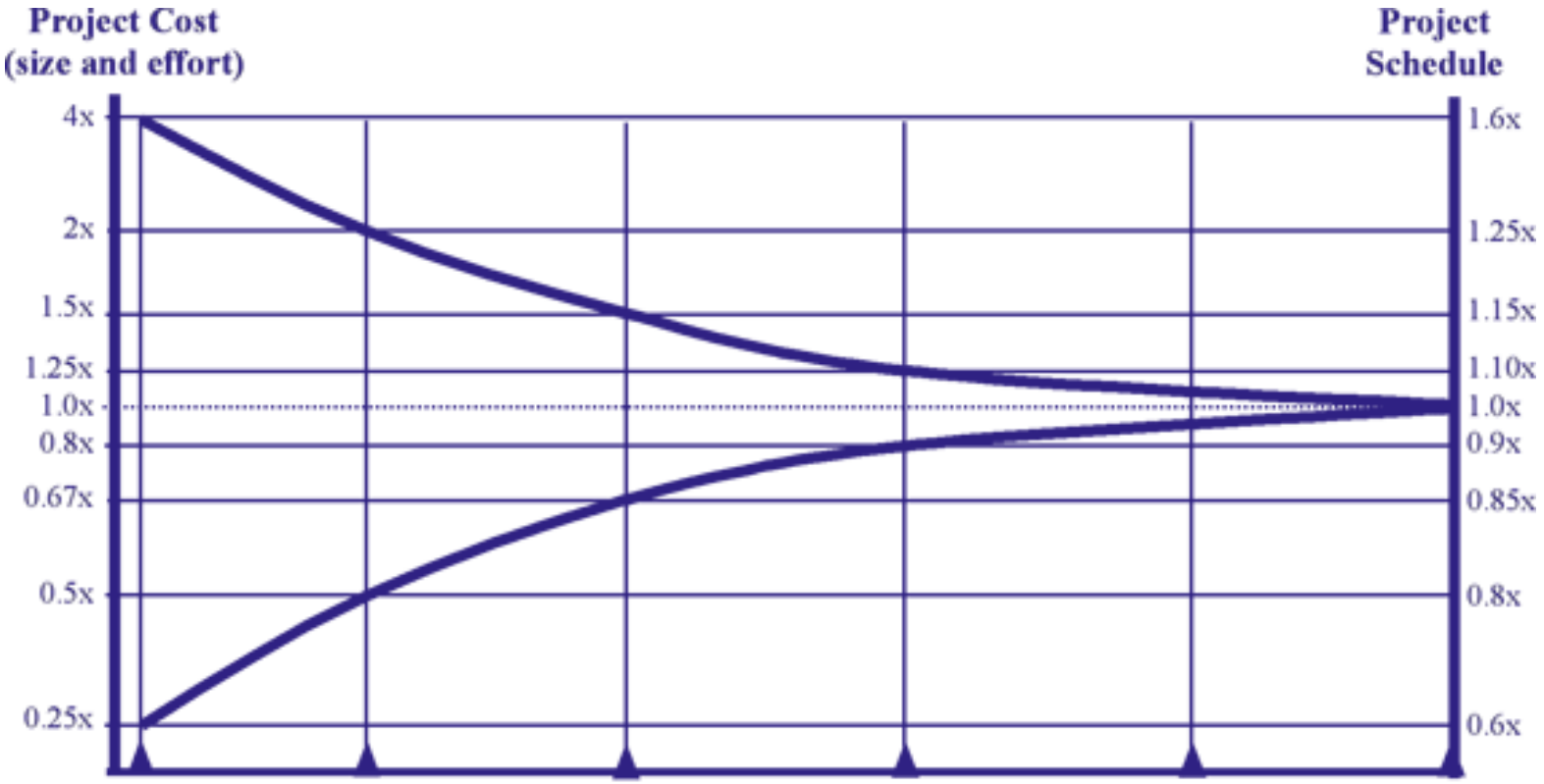
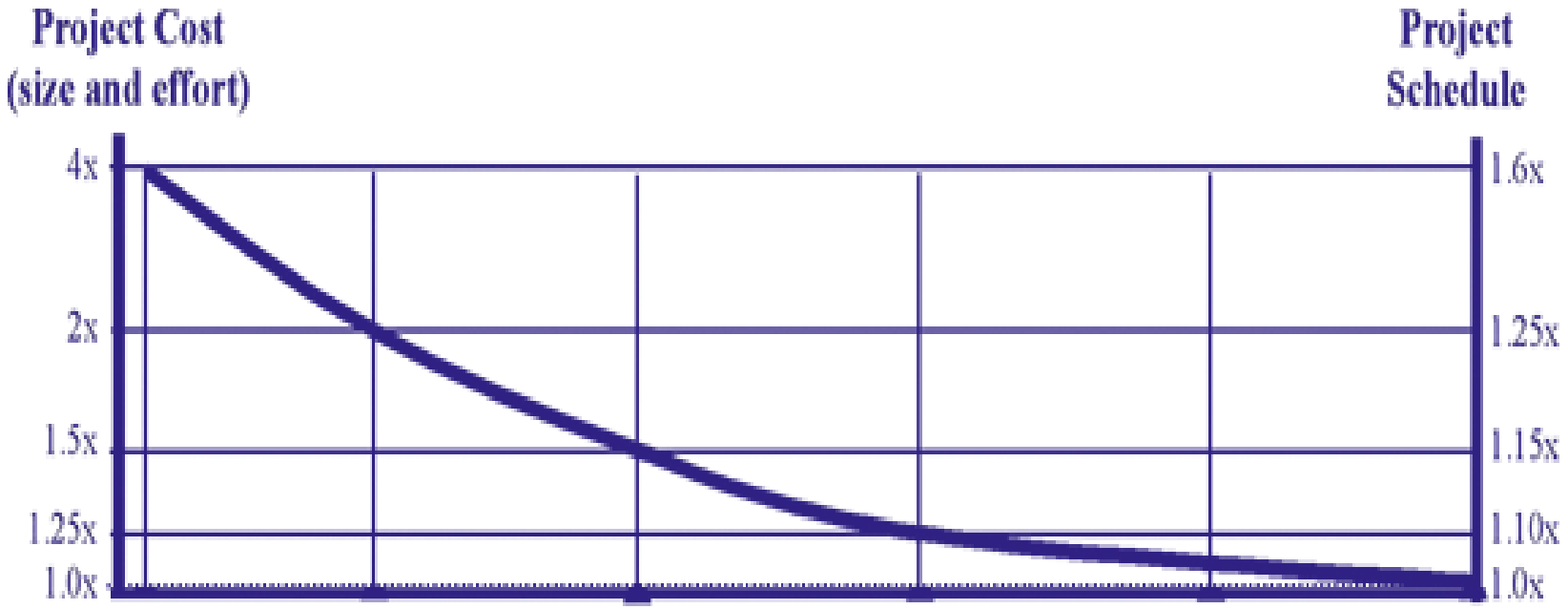
Why do we estimate?

How do we estimate?

The cone of uncertainty



The revised cone of uncertainty



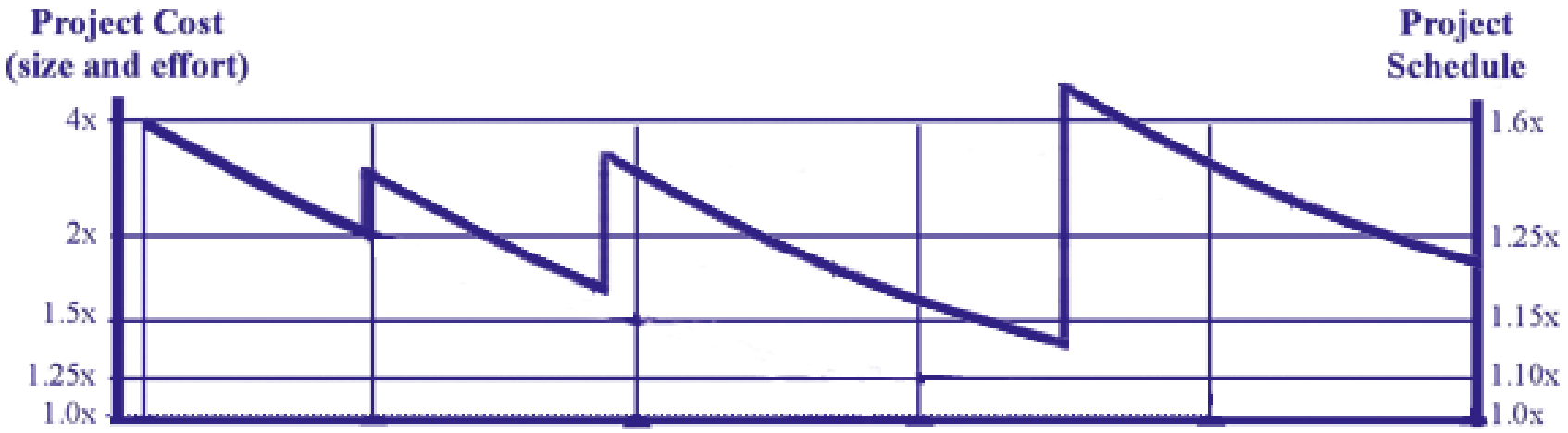
Why we will always underestimate

- Optimism Bias
- Planning Fallacy
- Cognitive Bias in general
- Hofstadter's Law

"It always takes longer than you expect, even when you take Hofstadter's Law into account.“

The real cone of uncertainty

(no assumptions)



Where do we live?

“Mediocristan”

- Height of human beings
- IQ
- Manufacturing a car

“Extremistan”

- Google search results
- Wealth of individuals
- Software development

Black Swans

"Unknown unknowns"

Price of oil
Dot com bubble
Credit crunch
9-11
Realising our definition of done was not really "done"
Not realising there would be so many difficulties with our hosting provider
Bugs

Scrum Estimating and Planning

- General estimation
  - T-shirt sizing (S, M, L, YGBK)
  - Complexity points (1, 2, 3, 5, 8, 13, 100)
  - Should never estimate in “man hours”
- Estimates made by the team doing the work
- Do some work to find out the velocity of the team
- Add range of error (e.g. Cone of uncertainty)

Agile methodologies do not make us more predictable

Teams get better at estimating as they go but only up to a point

- not necessarily any better than if we'd spent a lot of effort making fine grained estimates
- may be a lot worse to begin with

Working iteratively makes us more able to respond to uncertainty but does not mean we become any more predictable

A dose of reality

Teams change, requirements change, environments change

T-shirt sizing and story points get turned into man hours

People make commitments either explicitly or implicitly

Customer doesn’t care about assumptions, they want to know when it’ll be ready

Example 1

Sprint 0: no estimate

lets do some work and find out our velocity

Sprint 1: 3-5 months

Sprint 2: 4-6 months

Sprint 3: 4-7 months

Sprint 4: 5-7 months

team has really bedded in by this point and is making better estimates

Sprint 5: 6-7 months

Customer makes some implicit commitments on project being delivered in 4 months (allowing contingency of 1 month)

Sprint 7: 6-10 months

At this point a “Black Swan” event occurs – we find out the hosting environment does not have security compliance. New compliant kit has to be spec’d and built and all the software has to be moved and tested.

Example 2

Sprint 0: no estimate

lets do some work and find out our velocity

Sprint 1: 2-4 months

Sprint 1: 3-7 months

Sprint 3: 5-7 months

team has really bedded in by this point and is making better estimates

However, the BA leaves as s/he's been offered a load of money by Google

Sprint 4: 5-8 months

Customer makes some implicit commitments on project being delivered in 4 months (allowing contingency of 1 month)

Velocity drops as team realises what a contribution the BA made

Sprint 5: 6-10 months

Velocity drops even further as new BA has no idea what they were talking about

One of the developer's leaves so velocity drops yet again.

What's more important? What we do or how predicable we are?

Dr Dobbs' Journal did a survey on how we define success. They found:

- 61.3 percent of respondents said that it is more important to deliver a system when it is ready to be shipped than to deliver it on time.

- 87.3 percent said that meeting the actual needs of stakeholders is more important than building the system to specification.

- 79.6 percent said that providing the best return on investment (ROI) is more important than delivering a system under budget.

- 87.3 percent said that delivering high quality is more important than delivering on time and on budget

In chapter 4 of *Peopleware by* DeMarco and Lister they discuss a study done in 1985 by researchers at the University of New South Wales. The study analyzed 103 actual industrial programming projects and assigned each project a value on a “weighted metric of productivity”. They then compared the average productivity scores of projects grouped by how the projects’ estimates were arrived at. They found that programmers are more productive when working against their own estimates as opposed to estimates created by their boss or even estimates created jointly with their boss

The study also found that on projects where estimates were made by third-party system analysts the average productivity was even higher. This last result was a bit of a surprise, ruling out the theory that programmers are more productive when trying to meet their own estimates because they have more vested in them.

But the real surprise was that the highest average productivity was on those projects that didn’t estimate at all.