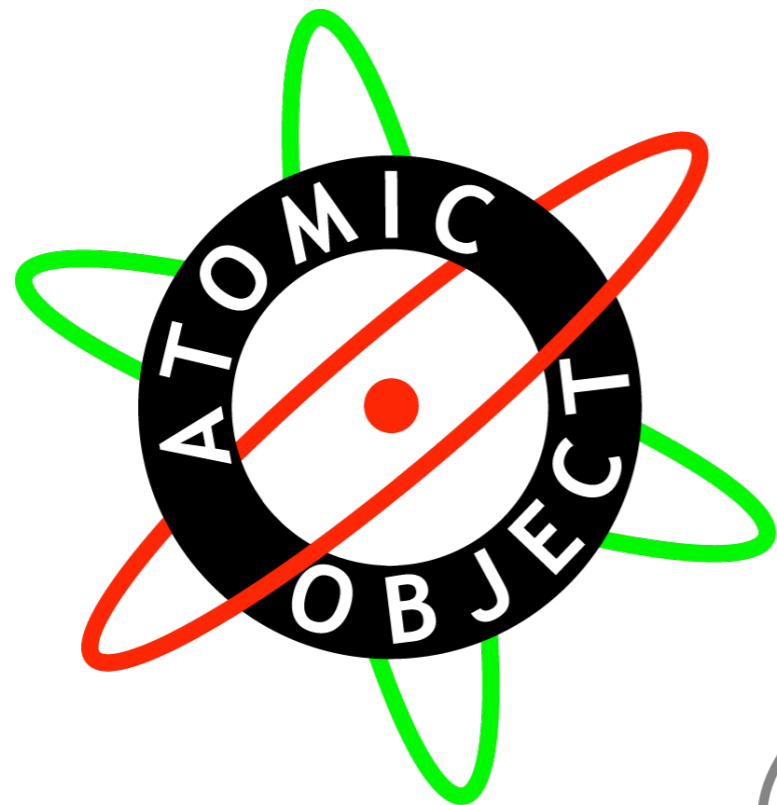




Convincing the Executive



Carl Erickson, PhD
Atomic Object

November 2007

(Click here to start)

Click on a link below to follow the slides for that idea.
Follow these links  to move to the next slide, and
the green arrows  to return to this index.

- Early delivery
- Timing and cost of bugs
- Fixing scope up-front
- Cost of maintenance
- Team size
- Killing your company
- Theory of Constraints
- Predictability
- Real Options
- Break the batch
- Eliminate waste

Theory of Constraints

- Eli Goldratt, Goldratt Institute, The Goal
- Applying TOC
 - identify the constraint
 - exploit the constraint
 - subordinate the rest of the system to the decisions above
 - elevate the constraint



TOC and Agile

- Favor generalists over specialists
 - simplifies the “factory”
- Programming becomes the constraint
 - exploit the constraint
 - ✓ connect the team to the customer
 - ✓ reduce waste via TD
 - ✓ fewer handoffs, less info loss



Happy Results

- Better utilization
- More market demand
 - predictability
 - quality
- Higher profitability
- Elevate via growth



Options - Background

- Financial Options
 - puts, calls, futures, derivatives, ...
 - can easily be sold
 - complicated models for pricing



Real Options

A real option is the right, but not the obligation, to undertake some business decision in the future



Decision Making

Defer decisions to the last
possible, responsible
moment



Agile and Real Options

- Automated regression test suites
 - change the software without fear
- Mock objects and TDD
 - defers design commitment
- Pairing
 - all parts known by at least 2 people
- Backlog and story-driven dev
 - defer decision on what to build

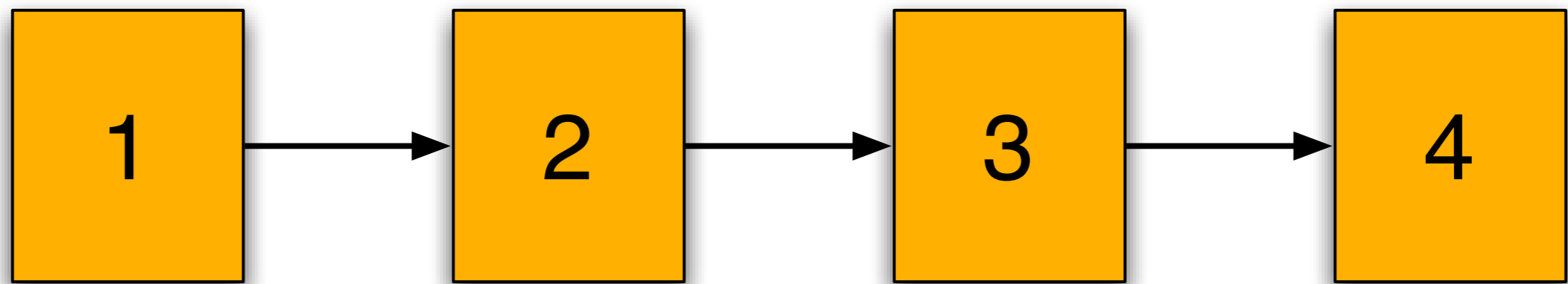


Using Real Options

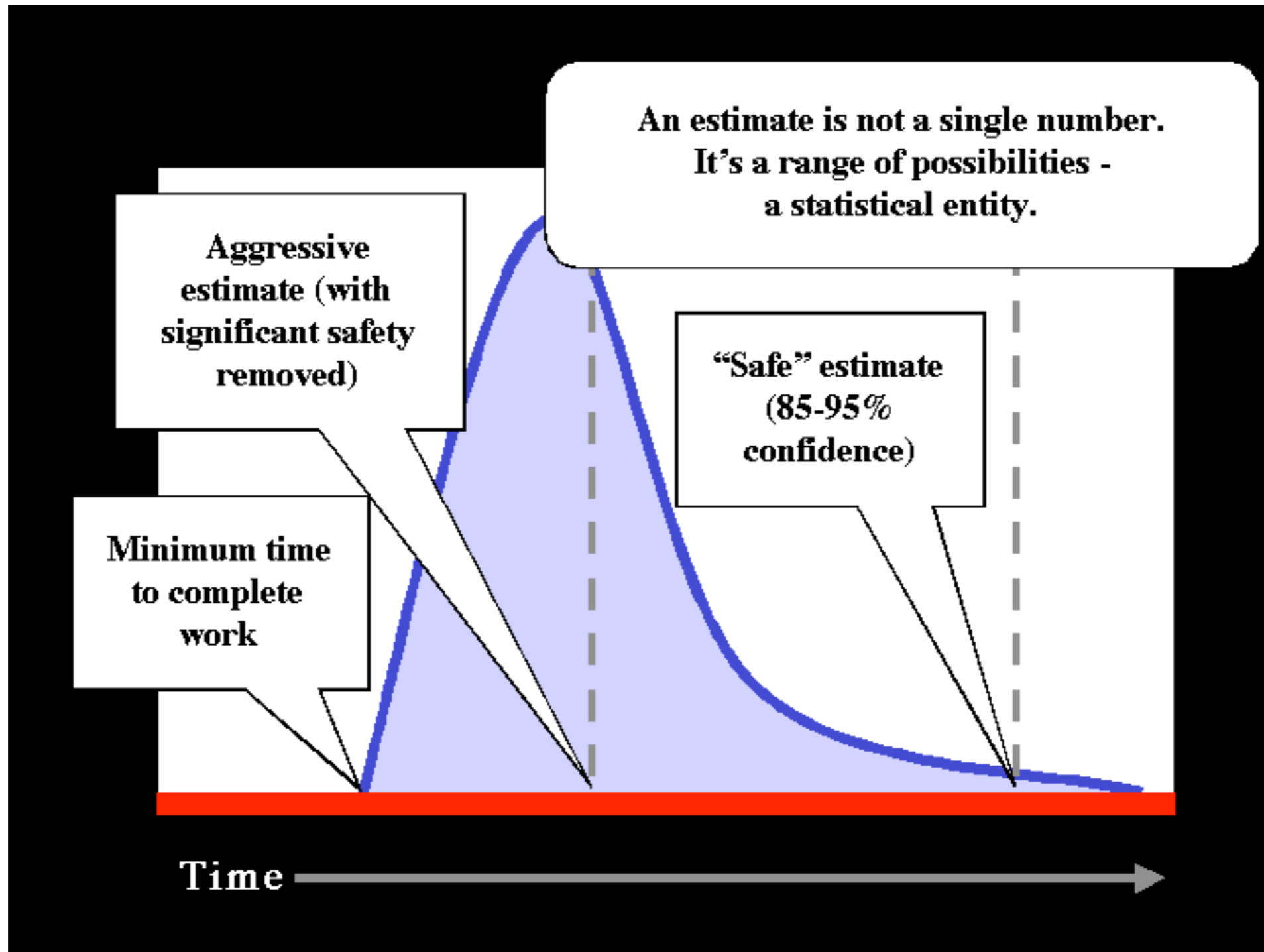
- Budget by iteration, rather than plan
 - feed the projects with best ROI
 - dump those not going well
- Staffing decisions
 - hold your best people to last
- Let customers drive projects
 - avoid building more than necessary



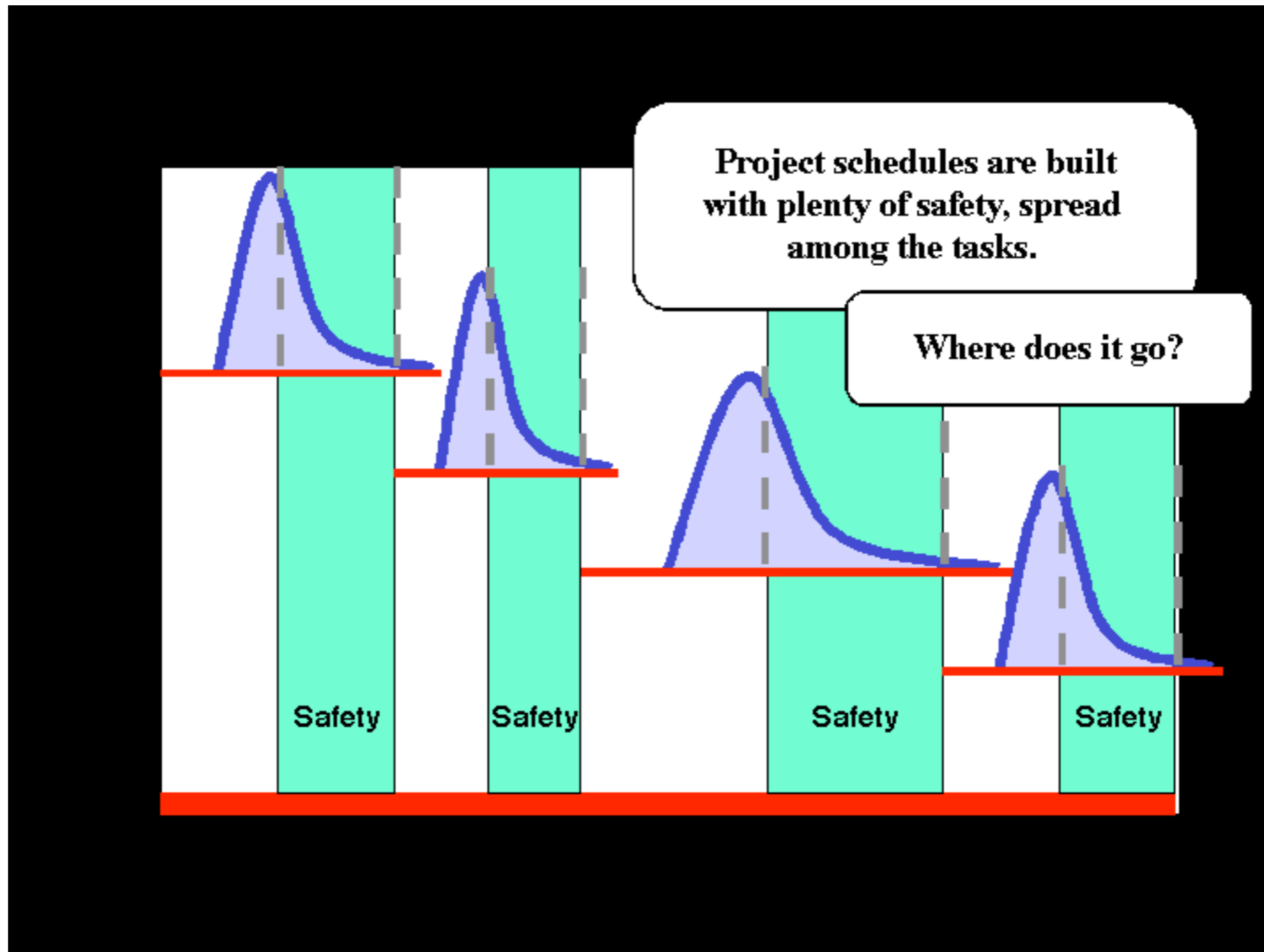
Predictability



PDF of Duration



Daisy Chain of Delay



Improving Predictability

- Significant business value
 - confidently making product plans
 - hitting revenue targets
 - executing on competitive strategy
- Agile practices increase predictability



Break the Batch

— requirements, design, program, test —



Break the Batch

~~requirements, design, program, test~~



Break the Batch

~~requirements, design, program, test~~

rd	rd	rd	rd	rd	rd	rd	rd	rd	rd
pt	pt	pt	pt	pt	pt	pt	pt	pt	pt

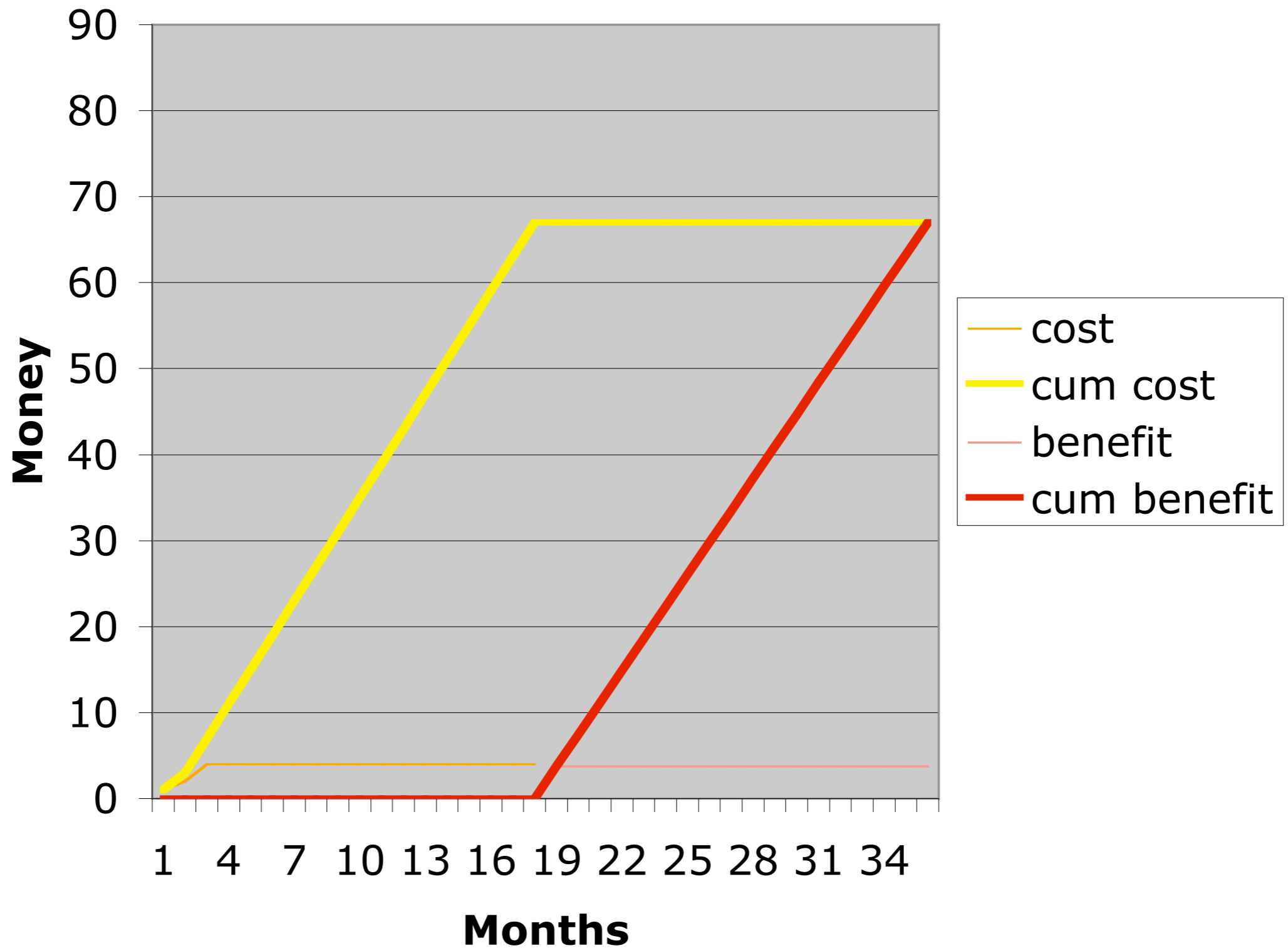


Early Delivery

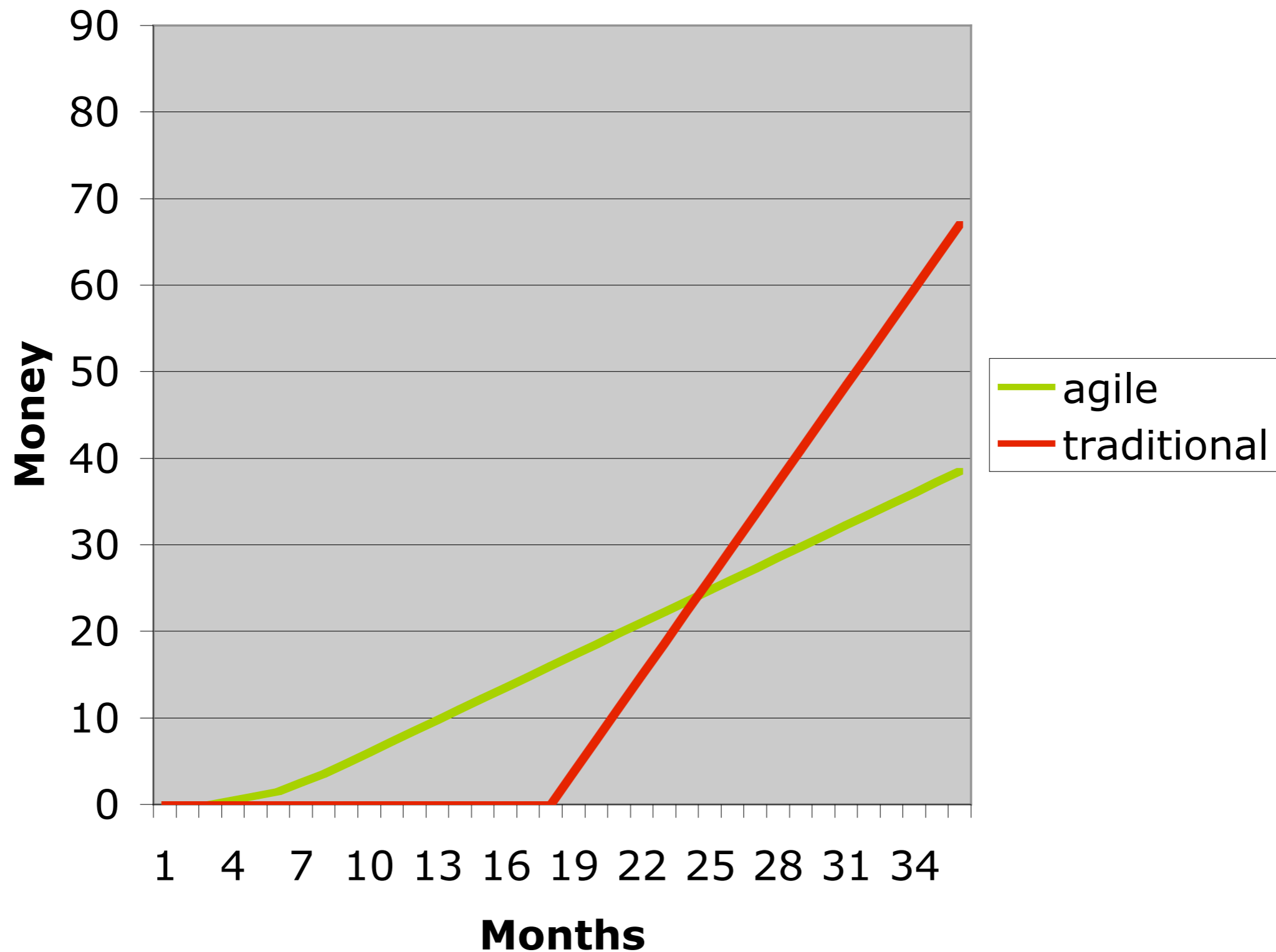
- Getting return earlier compounds in value over time (time-value of money)
- Reaching market sooner may have a huge impact



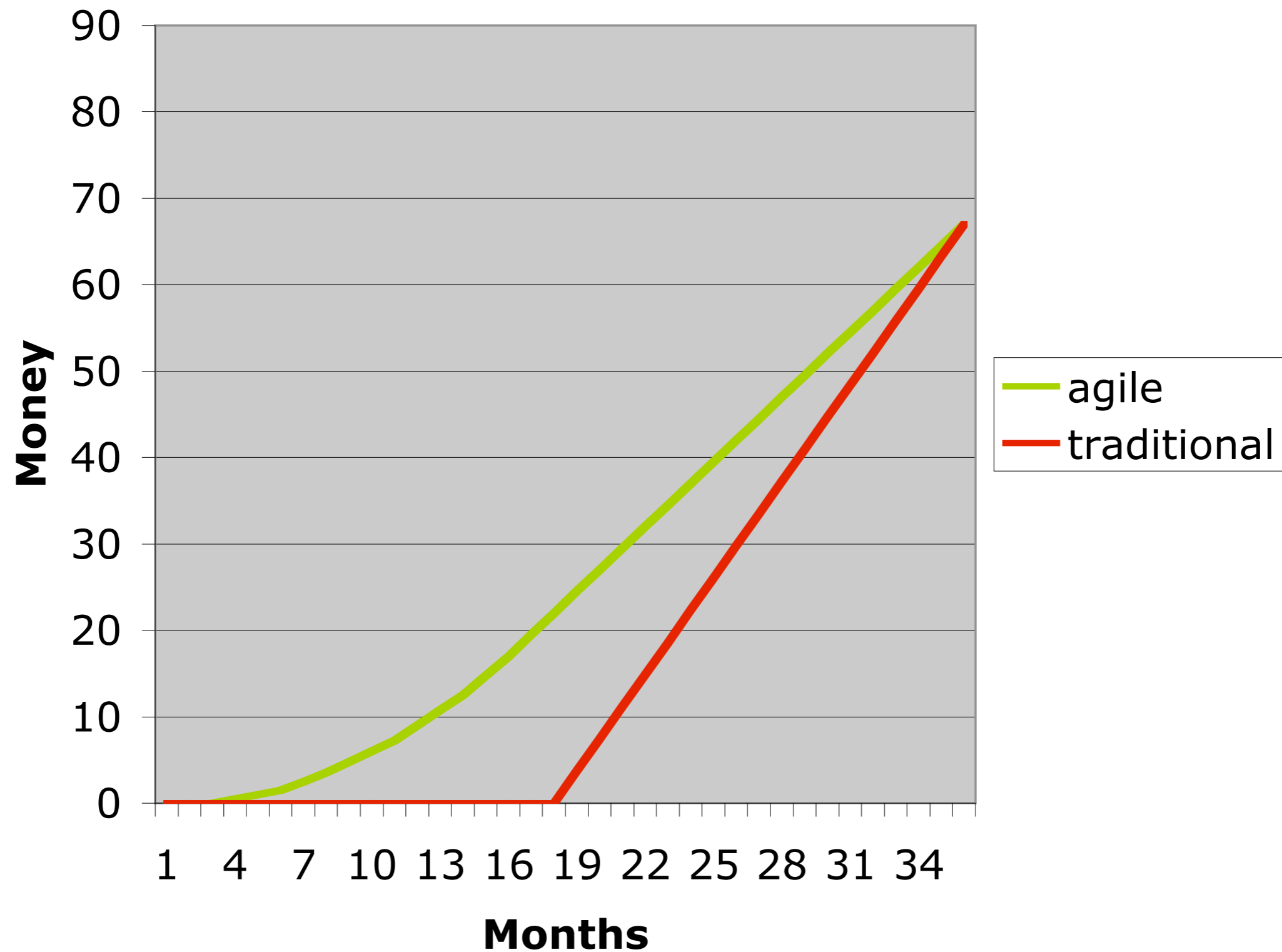
Cost and Benefit of Traditional Development



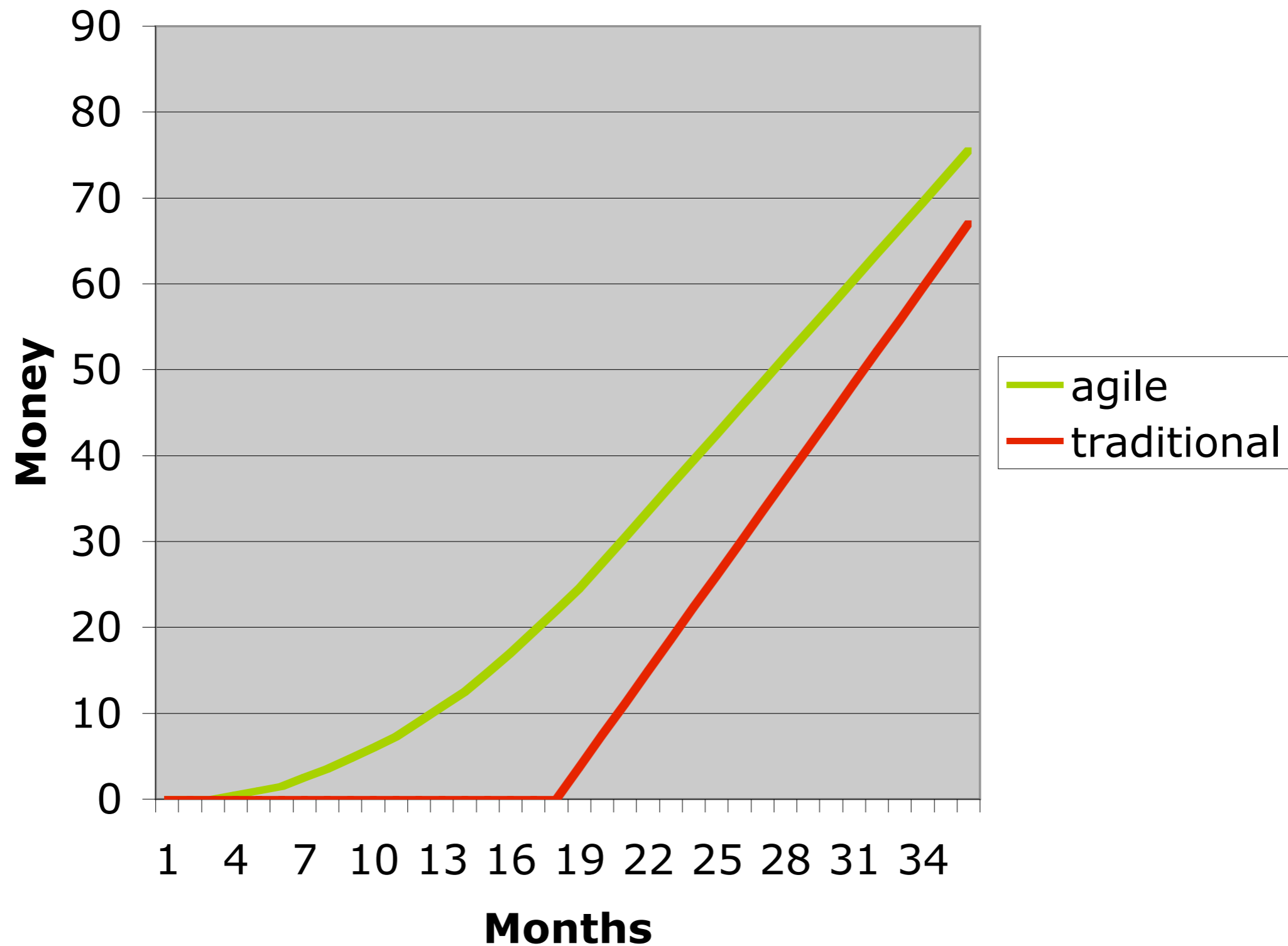
Traditional vs Agile (one project)



Traditional vs Agile (two projects)



Traditional vs Agile (two projects)

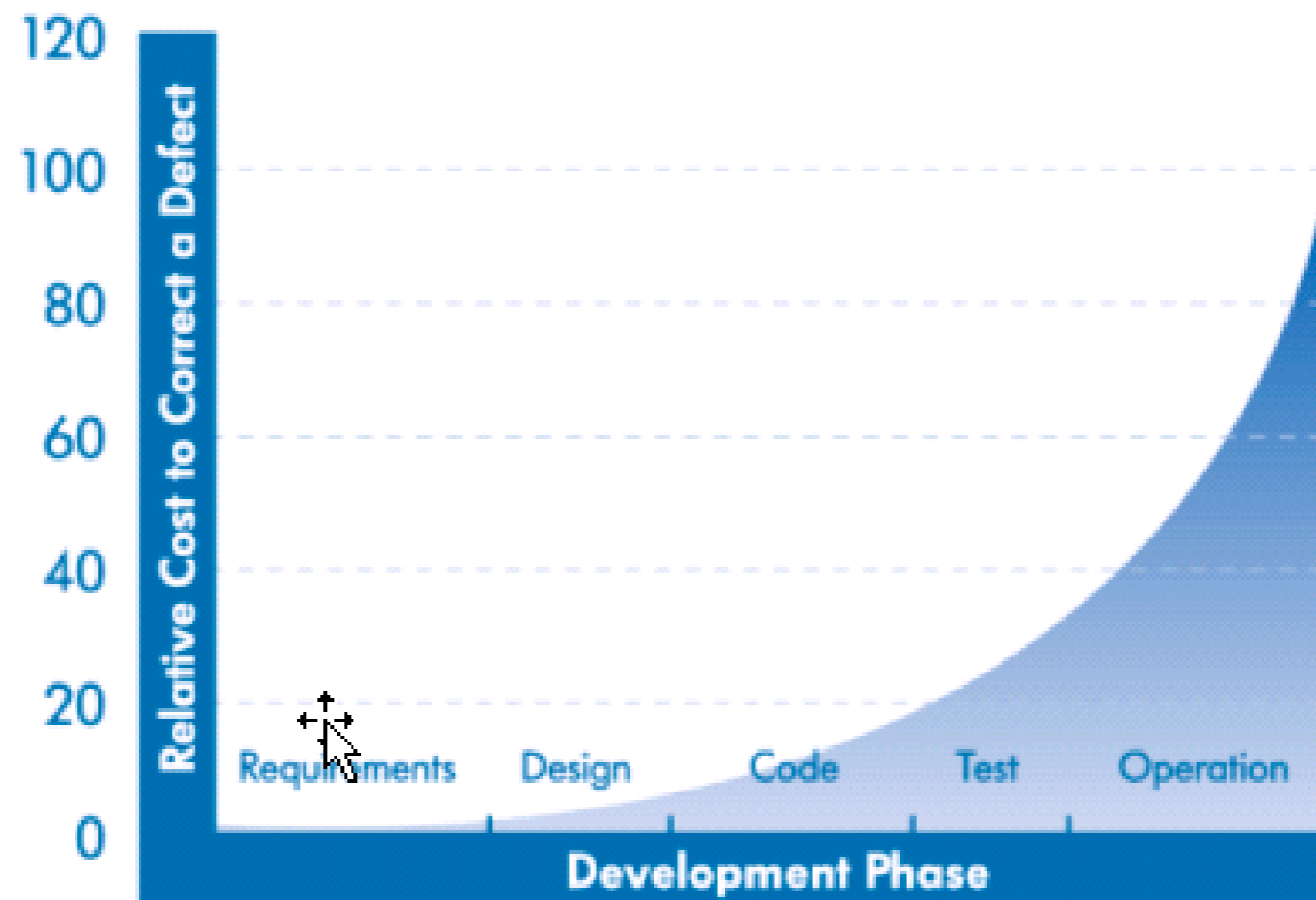


Fixing Scope Up-Front

- Change is expensive, so success comes from careful work in advance of coding
- Standish Study, 2002
- Key questions to ponder...



When You Fix Bugs



Doing it right the first time

- Test-driven development
- Automated testing
- Pair-everything
- Continuous integration
- Iterative, incremental delivery

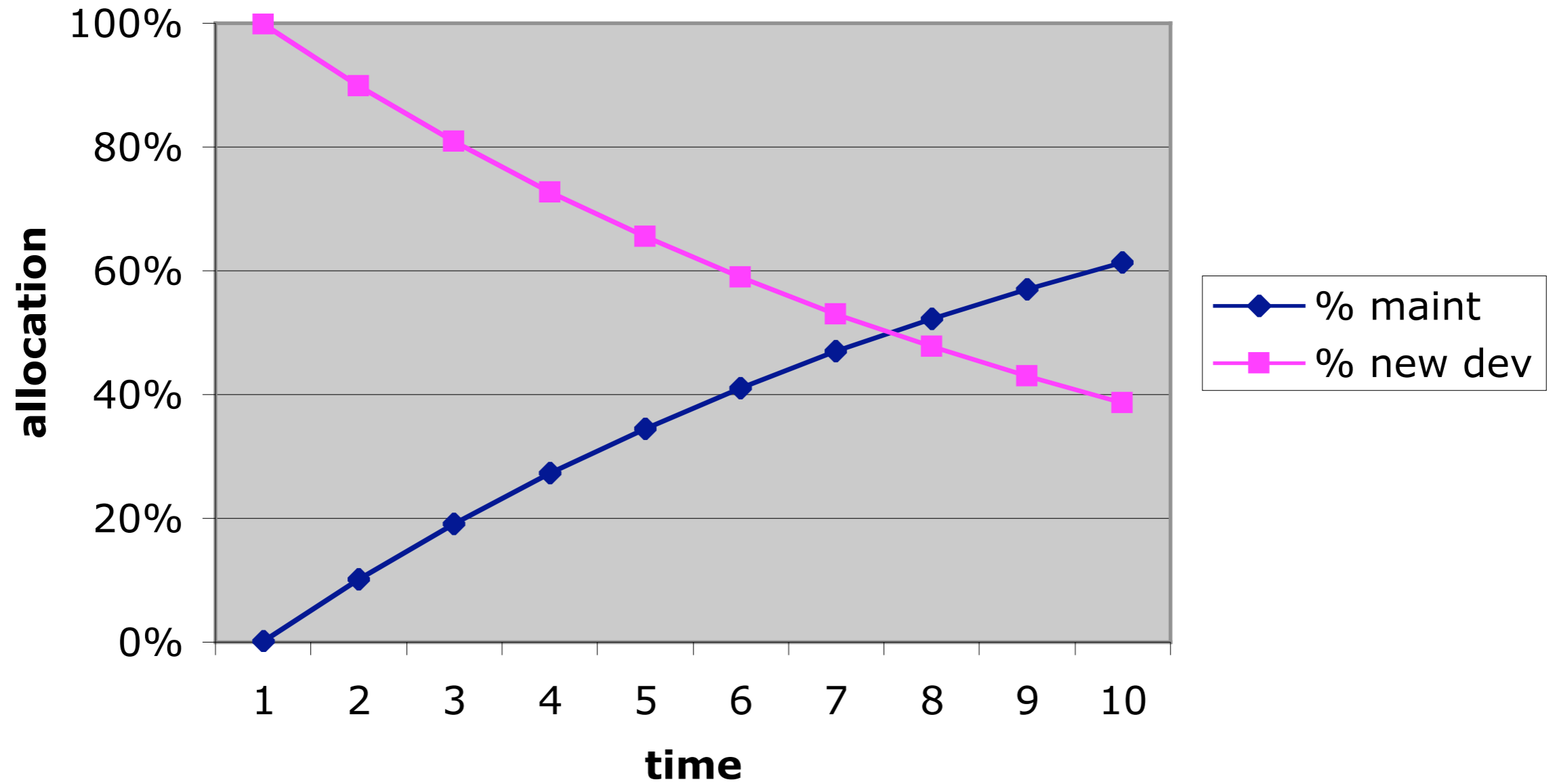


Cost of Maintenance

- Old apps never die
- Staff can't always grow
- Innovation gets squeezed



New Prod Dev vs Maintenance



Team Size

- QSM
 - Consultancy specializing in measuring, estimating, and controlling software dev
 - Database of 4000+ projects
- 2005 study on schedule vs team size
 - 564 information systems projects since 2002
- Divided into small (< 5) and large (> 20) projects by team size



Team Size, cont.

- For projects of 100,000 SLOCs
- Peak staffing of project
 - Average large team: 32 people
 - Average small team: 4 people
- Total effort for each?
 - 178 person months for large teams
 - 25 person months for small teams



Team Size, cont.

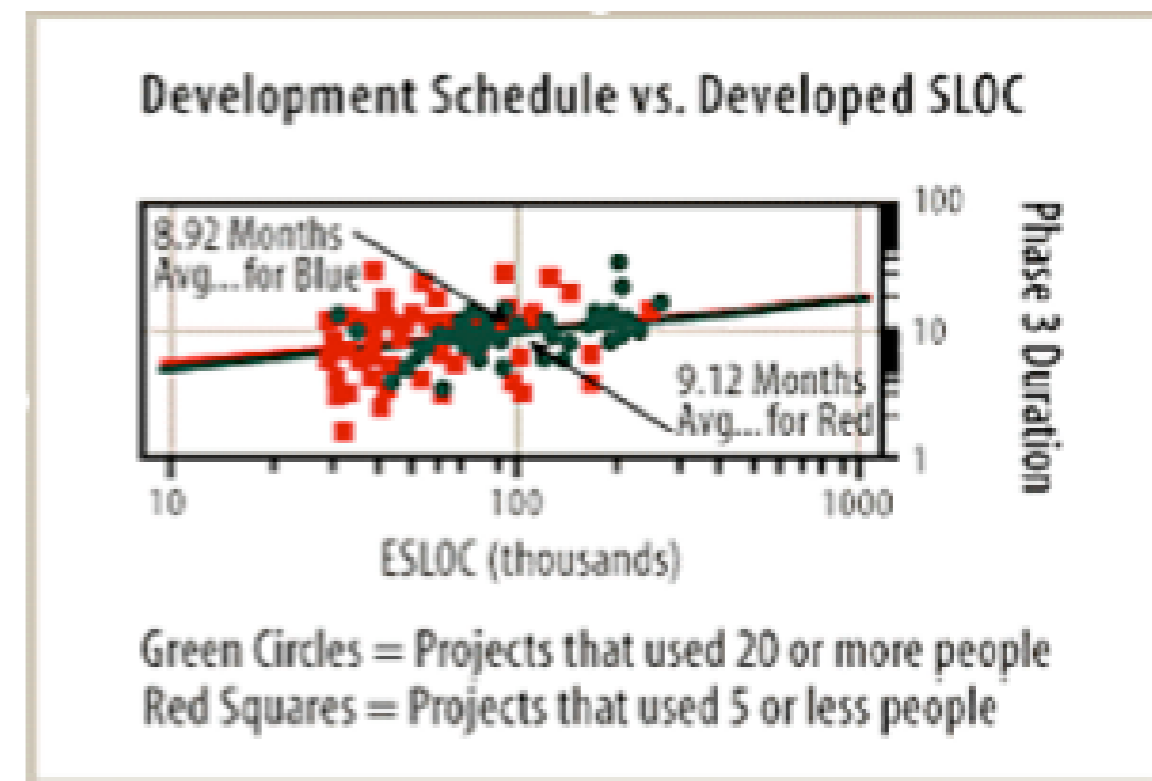
Did the big team finish first?

Was it worth the cost?

Explanations?

Communication and coordination
inefficiency

Greater rate of defects (5x)



Team Size, cont.

Did the big team finish first?

9.12 months for small team

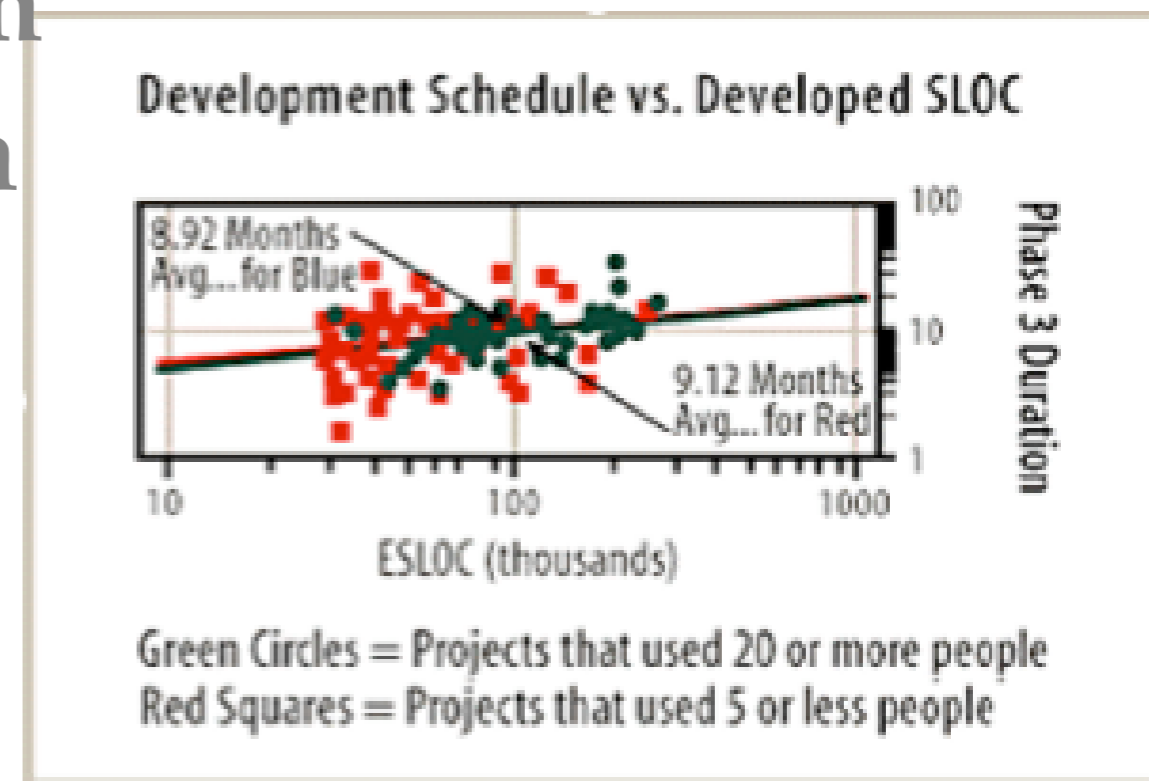
8.92 months for large team

Was it worth the cost?

Explanations?

Communication and coordination
inefficiency

Greater rate of defects (5x)



Team Size, cont.

Did the big team finish first?

9.12 months for small team

8.92 months for large team

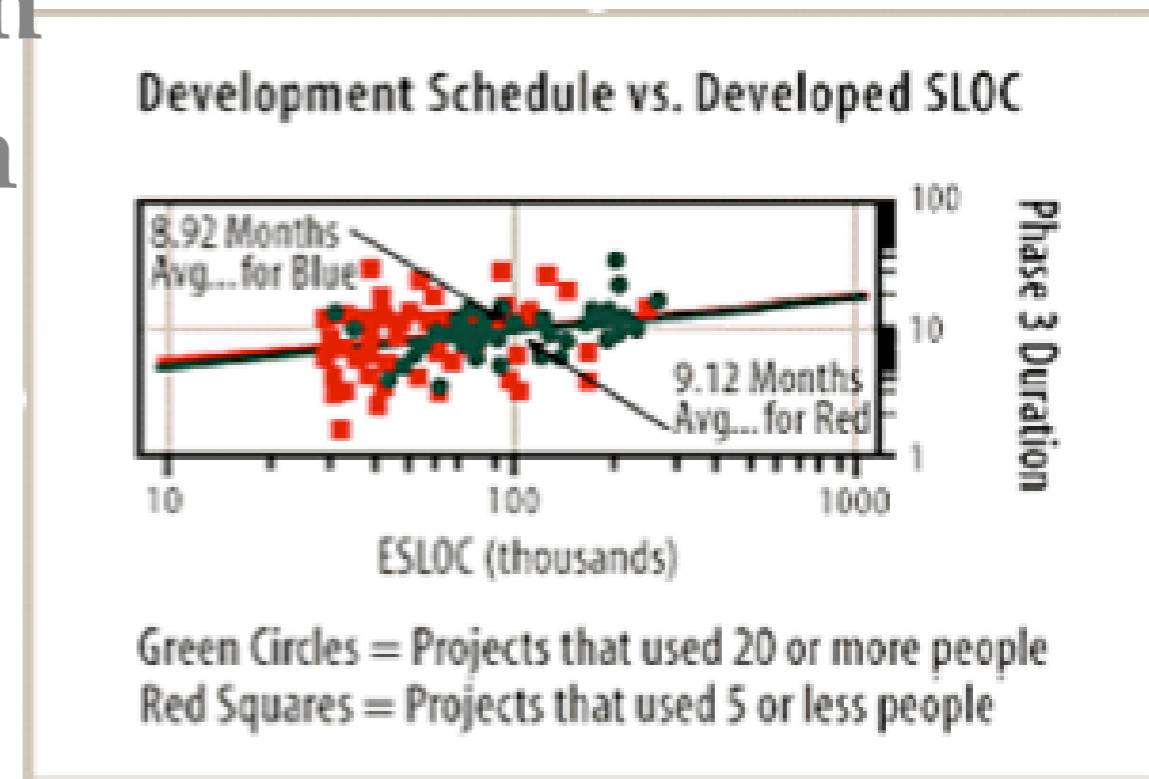
Was it worth the cost?

\$1.8M

Explanations?

Communication and coordination
inefficiency

Greater rate of defects (5x)

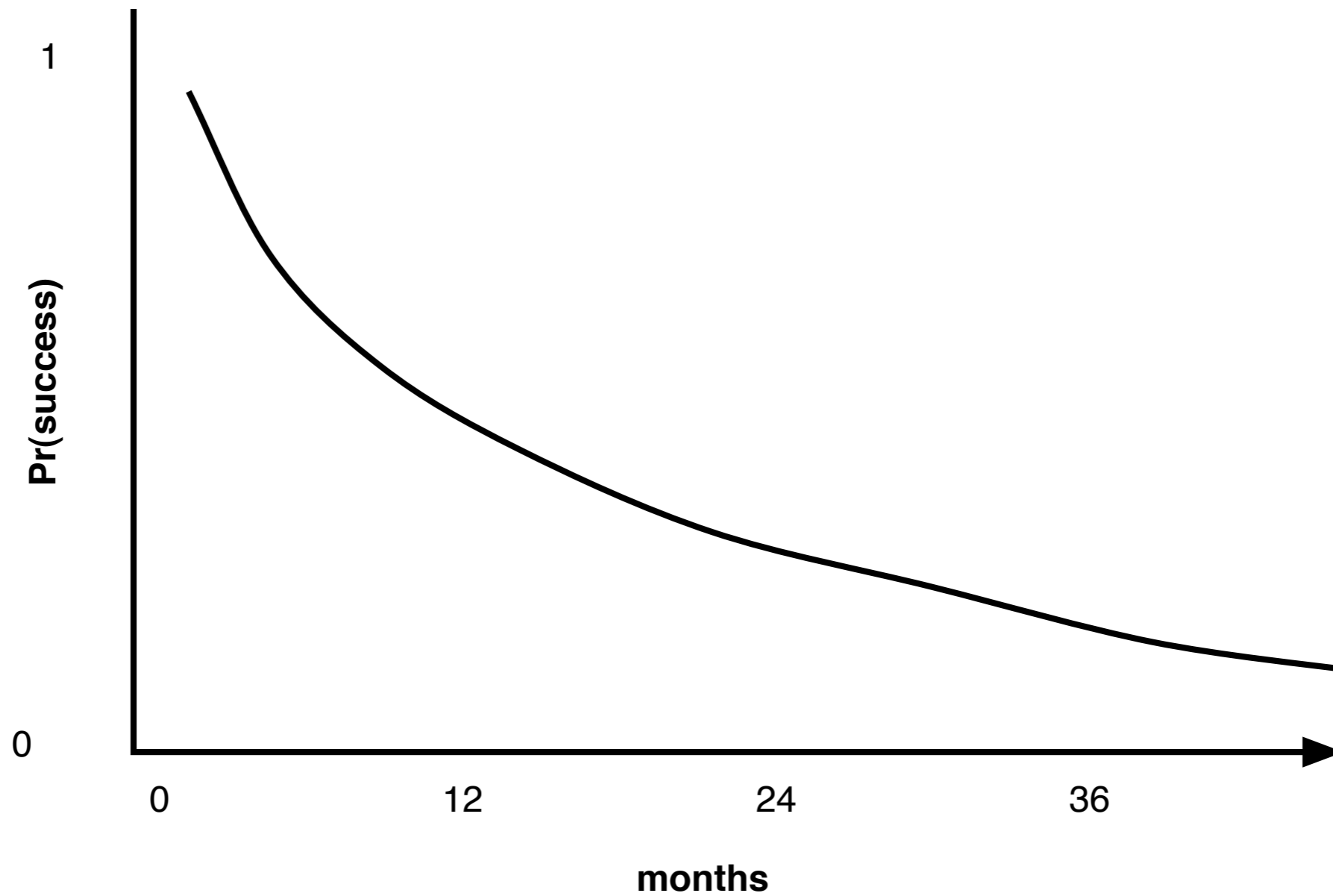


SSE Study

- Systematic Software Engineering
 - CMMi level 5
- Productivity of projects / teams
 - 1.8x small vs large (> 4000 hours)
- What can agile add to this?
 - productivity
 - better ROI



Project Size and Failure



Killer Core

- Traditional
 - $\text{value} = f(\text{time}, \text{quality})$
- Agile
 - $\text{value} = f(\text{time}, \text{quality}, \text{scope}, \text{cost})$



Belief in Magic

- Customer believes magic will happen (“we’ll deliver X by date Y”)
- Developers lie to their customers (“we can make it”), then cut quality to make it so



Core Functionality

- Aka “legacy”, “core”, “base”
- Velocity to work in this code is much lower
- How does this come to be?



Path of Doom

- New development
 - 6 months of work, velocity of 18
 - deadline is 5 months
 - quality cut to make the date
 - team is a hero



Subsequent Projects

- 6 months of work, 5 month deadline, velocity 17, cut quality, hit date
- 6 months of work, 5 month deadline, velocity 15, cut quality, hit date
- 6 months of work, 5 month deadline, velocity 10, cut quality, hit date
- ...



The Usual Signs

- Build core into each function
- Drop functionality
- Give developers to core teams to increase velocity
- Start rebuilding the core functionality
- Build new functionality and don't worry about core (fake "done")



The Meaning of Done

- Change the meaning of “done” to hit a deadline: *stabilization phase, user test, alpha, pre-release*
- Reduce quality with these common behaviors
 1. overtime and weekends
 2. cut testing
 3. cut reviews
 4. don't following standards
 5. no refactoring



The Death of a Company

- Ken says: 3-10 years for a company to back itself into a corner with their “core”
- Opens the door for your competition (innovative, not hobbled by slower dev)
- Cost of supporting magical beliefs is hidden
 - should be a top management decision
 - reflected on the balance sheet
 - every \$1 “saved” costs \$4 eventually



Eliminate Waste

manufacturing	software
overproduction	extra features
inventor	requirements
extra processing steps	extra steps
motion	finding information
defects	bugs not caught by tests
waiting	waiting
transportation	handoffs



software waste	agile practice
extra features	story-driven dev
requirements	story details
extra steps	story -> code
finding information	co-located team
bugs not caught by tests	TDD
waiting	deliver early, often
handoffs	developers <-> customers

