

# Comparison

Are you agile or fragile?

8/26/2023

vijaynathani.github.io

1

Swimming is a process even free-style swimming.

There is a time when panic is the appropriate response. - Eugene kleiner

=====

How did it get to be so wrong?

A short while ago I said that fixed-price contracts don't work. Over on the Scrum Development group there's a discussion about competitiveness, estimates and organization culture. Dave Martin said: *People underbid because it gets them the initial contract as many clients will just go for the lowest bid. Once the project is underway, the costs start to escalate and the client has 2 options*

- pay up or

- write the project off and start again.

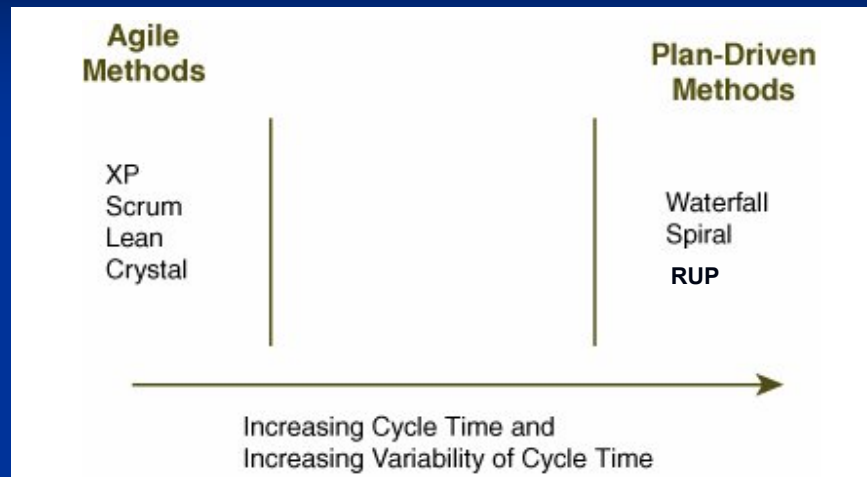
*The client is often better off writing off their initial investment and starting over but its amazing how often they don't do that and continue to burn money.*

Keith Sterling responded: *This is why so many large consultancies stick to the waterfall method. By bidding low and stipulating a waterfall approach, yet knowing that 99.99% of all projects will undergo 25-35% change during its lifetime, they know they will be able to make up the shortfall in their bid with high value change requests. It's a well known fact, yet unwritten rule that most large consultancies in the UK base their business model on the volume of change requests they can generate during a project, and why most of these organizations have some of the biggest legal departments I have ever seen.*

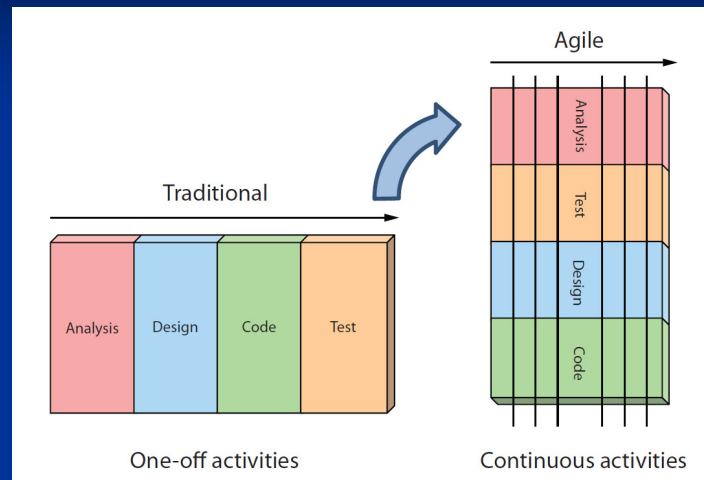
Where is the sense in awarding business based solely on the price? Price is meaningless without a measure of the quality being purchased. If clients award business to the lowest bidder they're likely to receive low quality and high cost. You get what you pay for. And the client-vendor relationship will become acrimonious as neither party is satisfied and it falls to the lawyers to fight it out. Deming predicated that *driving down the price without regard to quality and service will drive good vendors and good service out of business.* He was right but it's actually worse. As Keith describes, we now see companies making money from providing poor quality by charging extortionate sums for change requests. It's part of their business models. What a truly sad state of affairs.

Wouldn't it be better to enter into and nurture cooperative partnerships for the long-term that are built on mutual loyalty, trust and confidence, and which share the risks and the rewards? If you treat your partners as extensions to your business and align incentives so that everybody works for the good of the partnership, then quality will return, cost will fall, speed of delivery will increase, customers will be happy and everyone will realize prosperity.

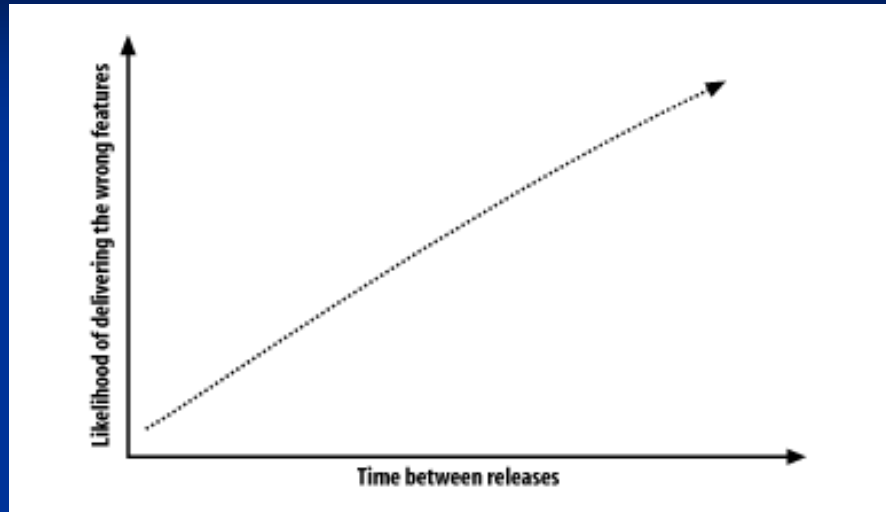
# Comparison



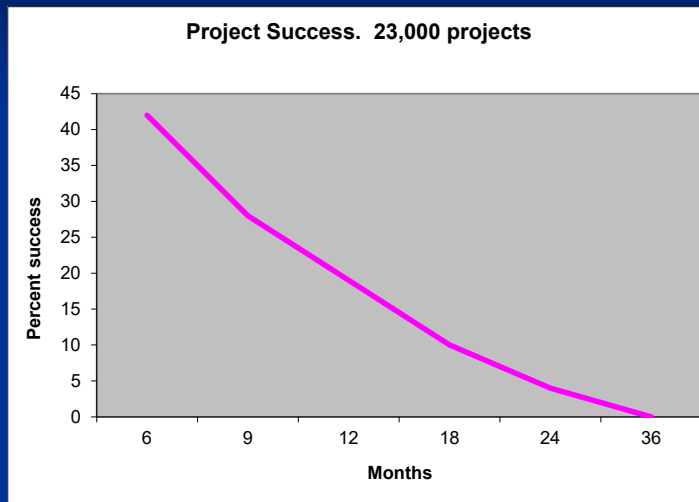
# Activities



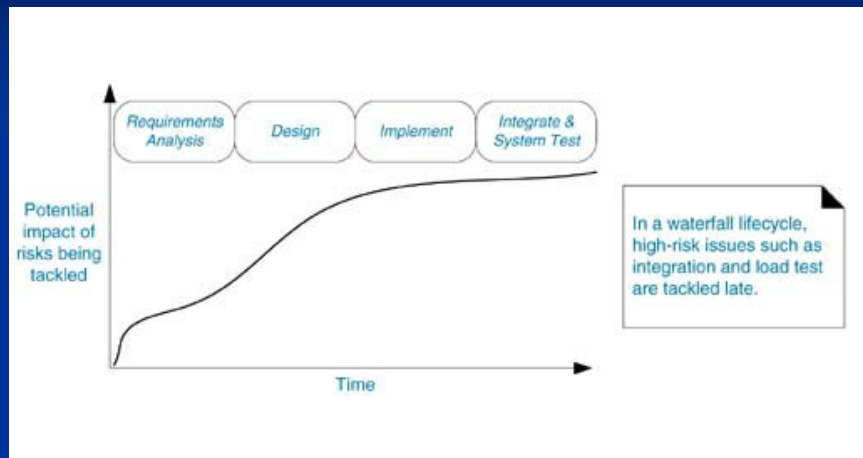
## Comparison



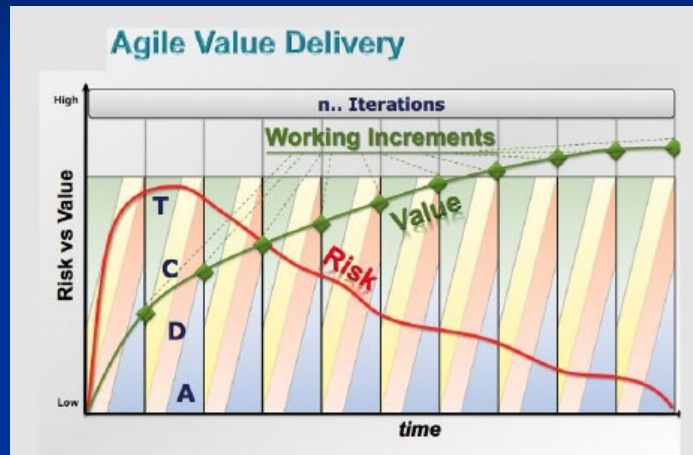
## Long Projects Fail.

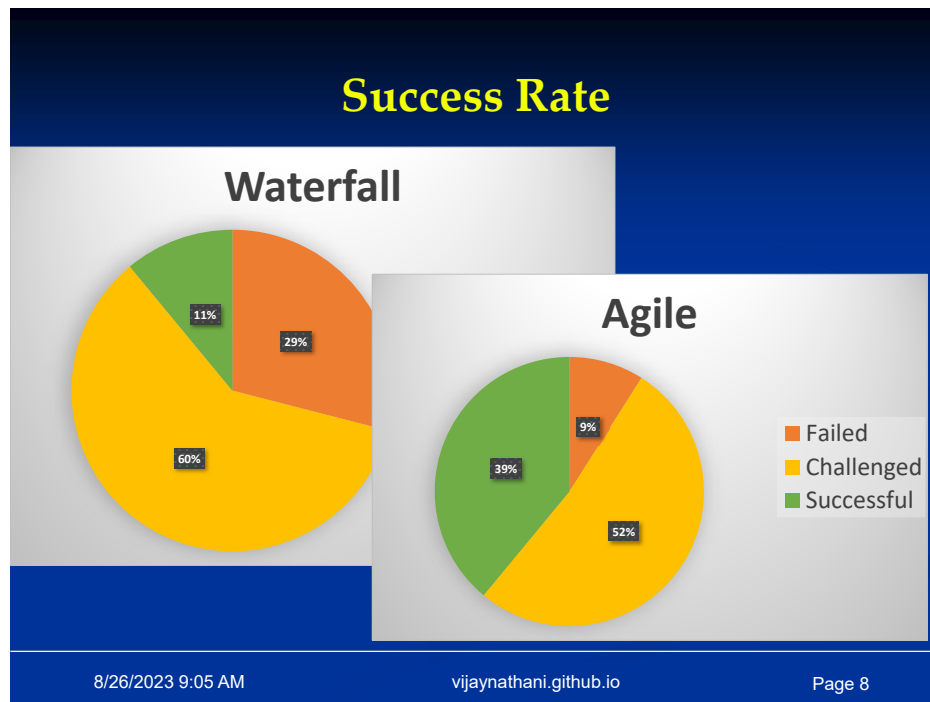


## Risk in Waterfall



## Risk / Value in Iterative



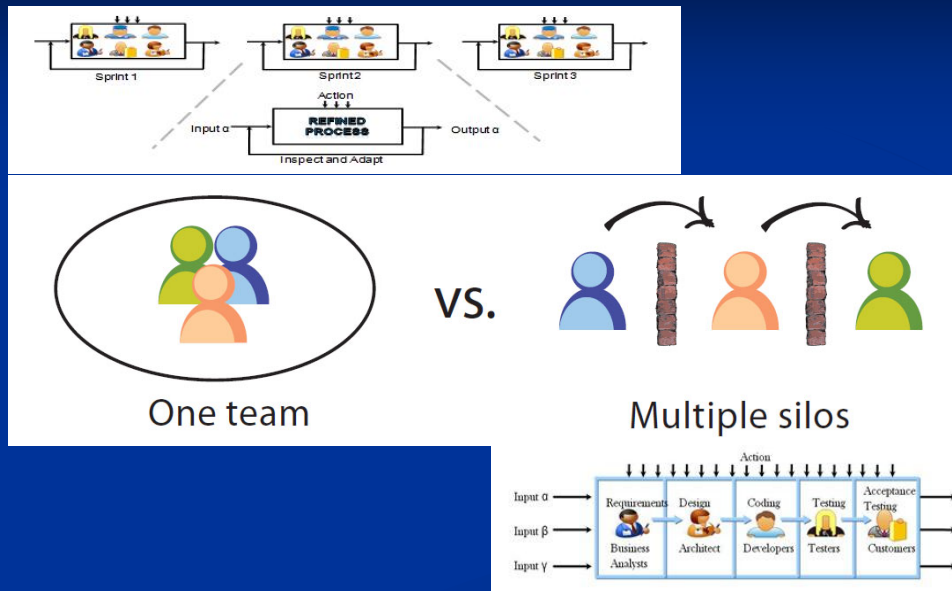


Standish Group Chaos Report 2015. Reference  
<http://www.infoq.com/articles/standish-chaos-2015>

More details in AgileVsTraditional.png



# Agile Team vs. Traditional Team

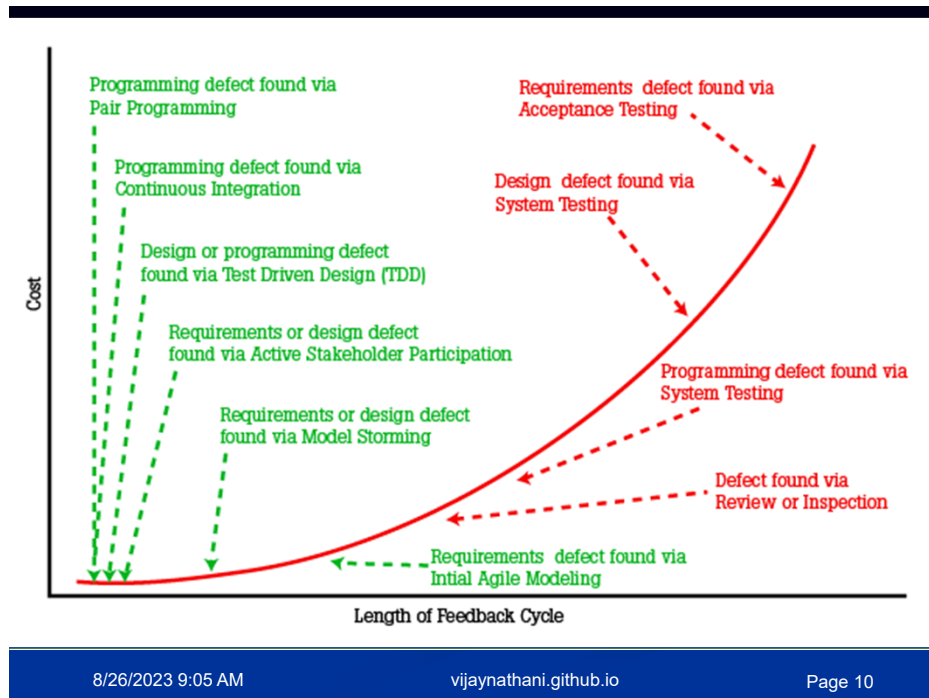


8/26/2023 9:05 AM

vijaynathani.github.io

Page 9

Silo – Pit dug for some specific purpose i.e. Separation  
No separate QA. Quality is everyone's responsibility.



Cost of change grows exponentially with time

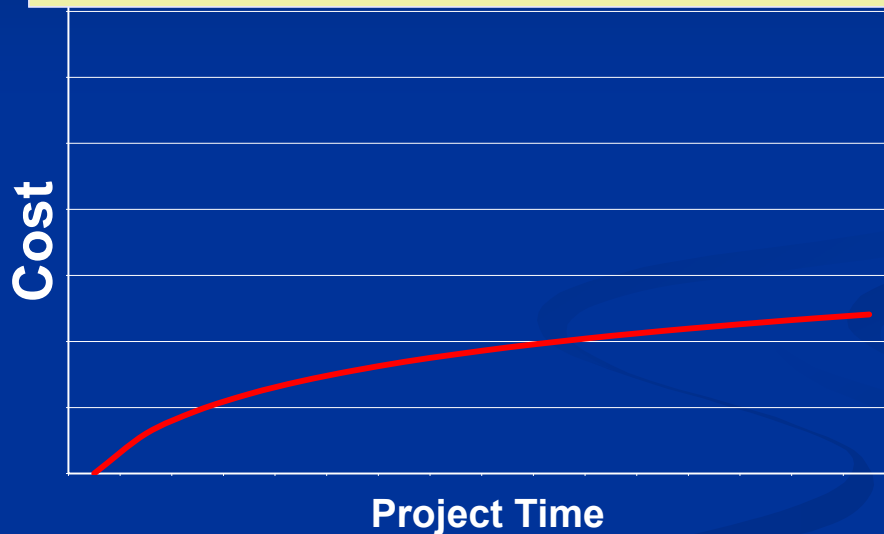
## A Financial Lingo Primer

One of the most important things that you can do as an IT professional is to learn and adopt common terms used by senior management and business stakeholders. Understanding these terms will improve your ability to communicate and will increase your appreciation of non-technical issues—the best developers know that there is more to development than development.

- **Diminishing Returns.** As more investment in something is made, the overall return on investment (ROI) increases at a declining rate until it reaches a point where it begins to decline. For example, if a diagram isn't yet good enough for the situation at hand, then you can keep working on it until it is good enough, at which point any more work on that diagram is a wasted effort. Although the continued work still adds value, it doesn't add as much value as when you first started because you likely addressed the critical issues right away. The concept of diminishing returns is critical because it enables you to recognize that it doesn't make sense to try to "complete" a work product.
- **Discount Rate.** The rate at which future cash flows are adjusted to reflect the time uncertainty of money—a dollar tomorrow is worth slightly less than a dollar today. Minimally the discount rate is at least the expected rate of inflation and it is required to calculate the NPV of a project.
- **Internal Rate of Return (IRR).** The discount rate that makes the project have a zero NPV. For a project, the IRR is equivalent to the interest rate at which you would need to invest your money to get the same value as a project. IRR is used to compare the value of projects. For example, say you have a project that initially cost \$500,000 and generated net benefits of \$300,000 in the first year, \$400,000 in the second year, \$250,000 in the third year, and \$100,000 in its fourth and final year of production the IRR is 45.3%.

# Agile

...for much of the life of the system.



So, perhaps the cost of change can be flat But a few things have changed in three decades.  
Agile does not have overhead of BRUP, BDUP. It normally uses TDD. Hence the graph changes.  
We don't have to walk down the hall and submit a deck of cards to the operator  
and then wait a day for our compile to finish  
Computers are 1000X faster and 1000X cheaper.

8/26/2023 9:05 AM

vijaynathani.github.io

Page 11

→ 1,000,000 X power/\$ !!

Compile/test cycle has gone from days to seconds.

We have CI and automated tests.

Compile cycles have gone down to minutes and seconds.

Finally, OO languages and principles make software much easier to change.

*If tools, practices, and principles are properly employed.*

When costs don't dramatically increase with time: Up front work becomes a liability, Ambiguity or volatility is reason to delay, It is cost effective to delay all decisions until the last possible moment.

We pay for up front speculative work, some of which will certainly be wrong.

So we don't plan for something that never happens.

We only pay for what we use.

If you implement a feature today, but it turns out not to be valuable, you lose money and opportunity.

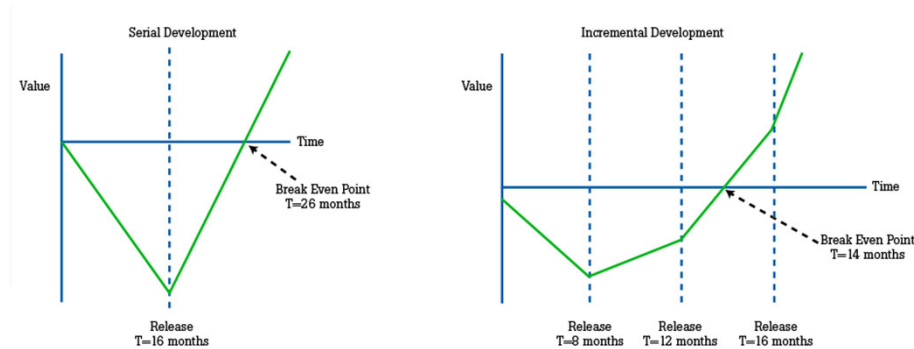
If you are uncertain and you can wait, then the risk goes away over time.

Time answers questions and removes uncertainty.

We need a process that creates and then exploits a flat change-cost curve.

XP is such a process.

# Cost Benefit for Waterfall vs. Agile



8/26/2023 9:05 AM

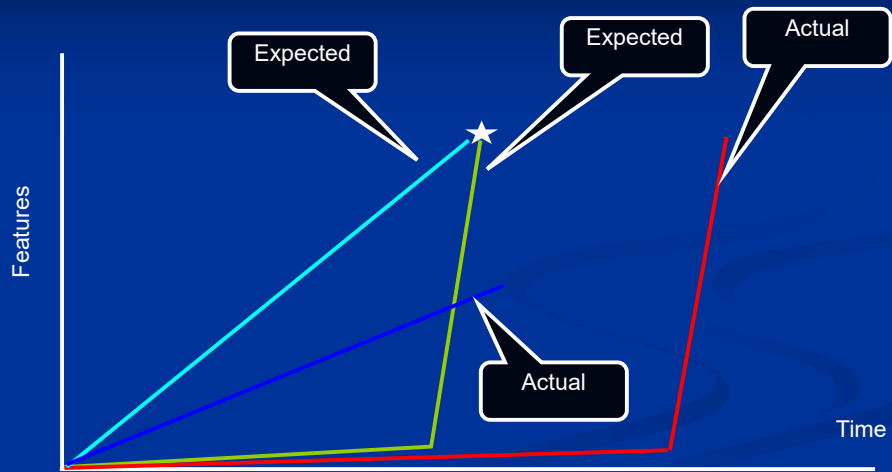
vijaynathani.github.io

Page 12

## A Financial Lingo Primer

- Net Present Value (NPV).** The NPV of a project is the sum of the present values of the annual cash flows (the revenues less the cost) minus the initial investment. The cash flows are discounted to take into account the time uncertainty of money. For example, with a discount rate of 10% per year, one dollar a year from now is only worth \$0.91 today ( $1/1.10$ ). NPV is a critical concept because it enables you to compare things that have varying initial costs and cash flows.
- Opportunity Cost.** This is the cost of passing up a choice when making a decision. For example, if you could have spent \$10,000 in additional testing and avoided a mistake that ended up costing you \$15,000 to fix, then the opportunity cost of not testing was \$5,000 (\$15,000—\$10,000). The concept of opportunity cost enables you to communicate the value in taking, or not taking, a course of action.
- Payback Period.** This is the length of time required to recover an initial investment through the discounted cash flows generated by the investment. The shorter the payback period, the lower the financial risk of a project.
- Return on Investment (ROI).** ROI is the number of times the net benefits (revenues minus costs) recover the original investment. The greater the ROI, the better the project. For example, assume a project initially costs \$500,000, generates revenues of \$250,000 a year with an annual operating cost of \$50,000, and runs for ten years. With a discount rate of 0% the NPV of the project is \$1.5 million  $((\$250,000 - \$50,000) \times 10 - \$500,000)$ , therefore, the ROI is 3. With a discount rate of 5% the NPV is \$1,044,347, therefore, the ROI is 2.088.
- Time to Market.** The length of time it takes to get a product from idea to marketplace. In the case of a software development project, the time it takes you to get your system into production. The longer the time to market, generally, the greater the risk.
- Total Cost of Ownership (TCO).** TCO is the total costs over the lifetime of a work product. TCO captures the true costs of an item, such as a document, component, or system, not just the initial investment. For example, for the aforementioned system, the TCO with a discount rate of 0% is \$1 million  $(\$500,000 + 10 \times \$50,000)$  and with a discount rate of 5% the TCO is \$886,087.

# Project Execution



How long did it take apps to reach 100m users:

LinkedIn: 7 years, 11 months  
Google Translate: 6 years, 6 months  
Pinterest: 5 years, 11 months  
Uber: 5 years, 10 months  
Twitter: 5 years, 5 months  
Telegram: 5 years, 1 month  
Spotify: 4 years, 7 months  
Facebook: 4 years, 6 month  
YouTube: 4 years, 1 month  
Snapchat: 3 years, 8 months  
WhatsApp: 3 years, 6 months  
myspace.com: 3 years  
Instagram: 2 years, 6 months  
WeChat: 1 year, 2 months  
TikTok: 9 months  
ChatGPT: 2 months

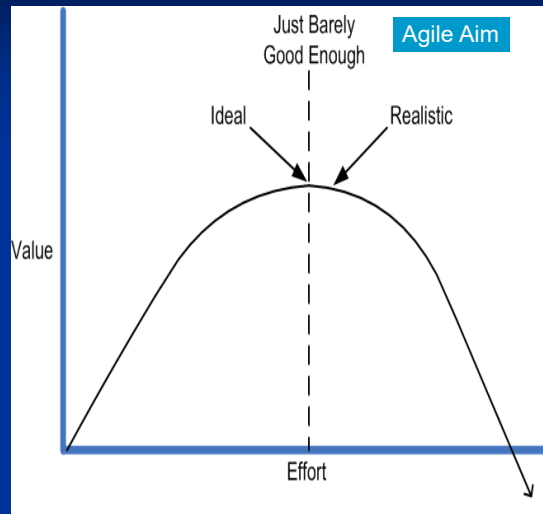
## Competitiveness – Accelerating Pace of change

For most companies: Time to  
market is more important  
than the cost

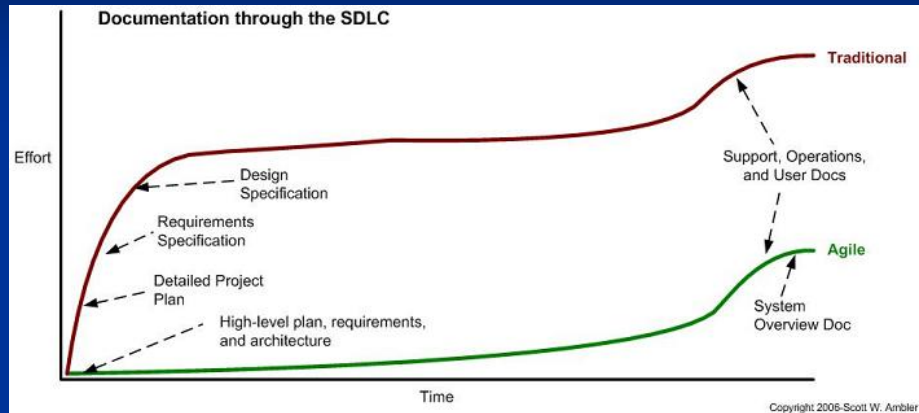
# Documentation



Traditional



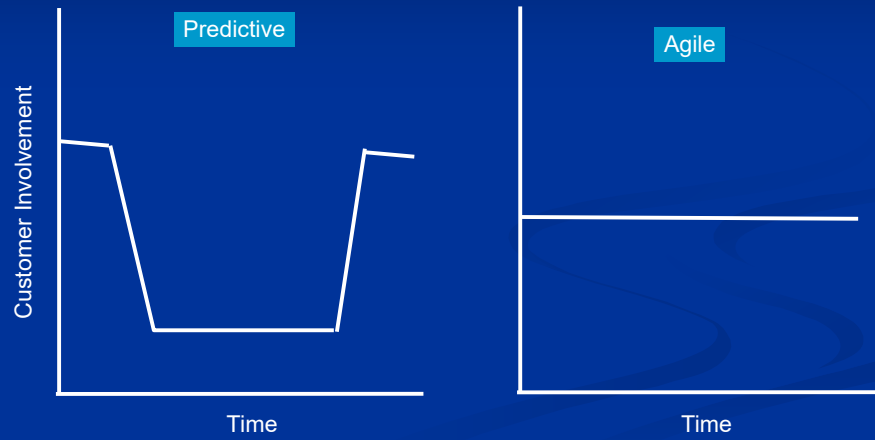
## Comparison of Documentation



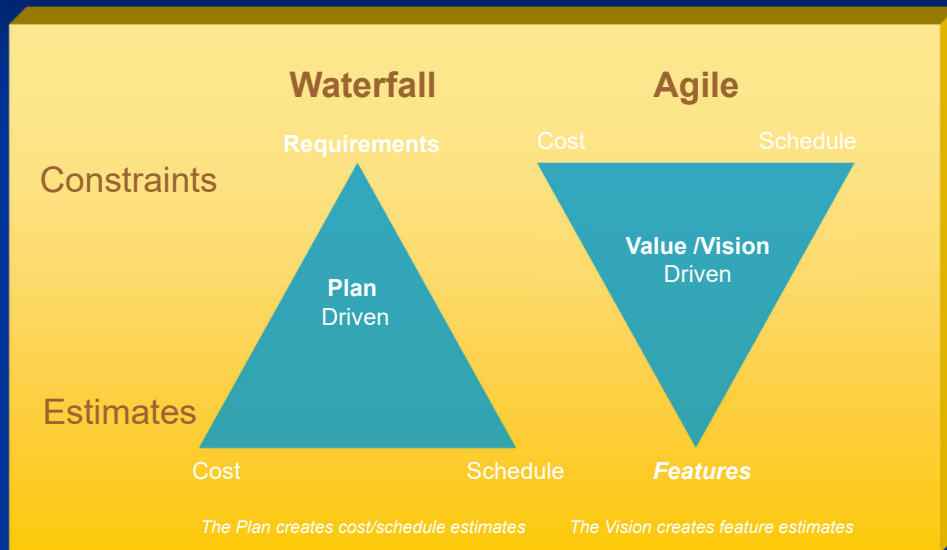
In Agile, documentation is typically 5% of waterfall.

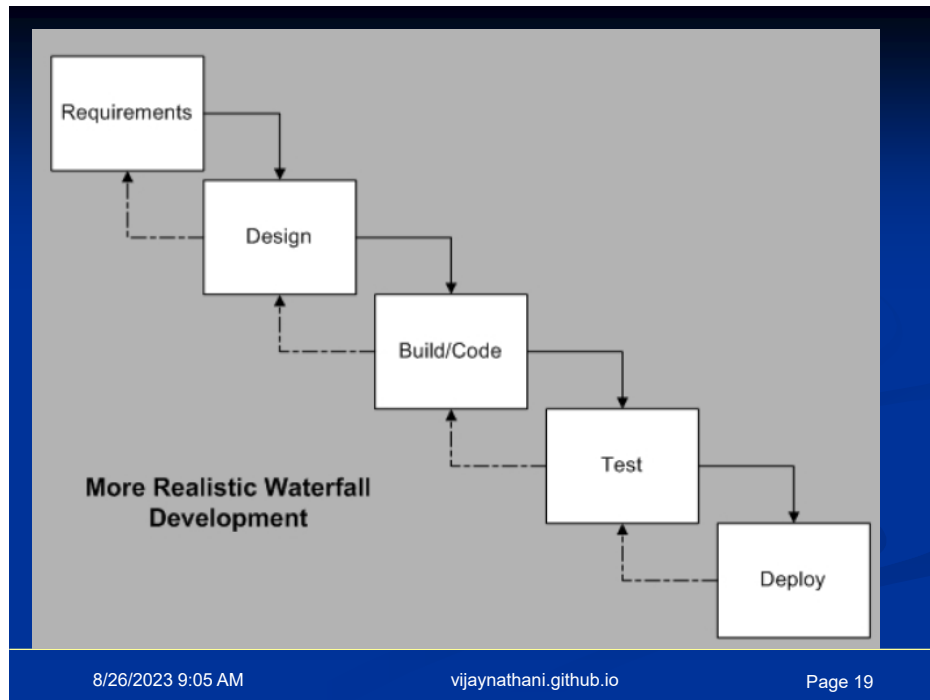


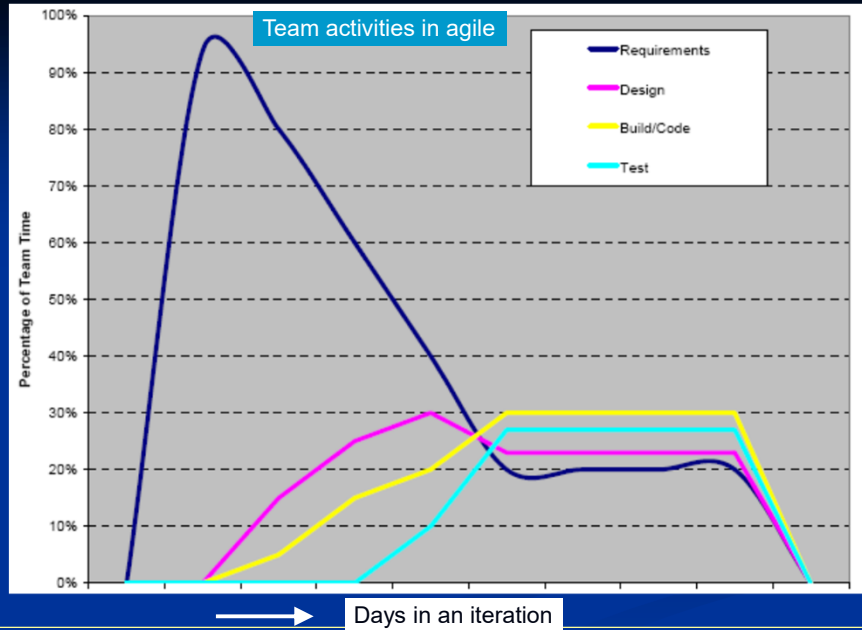
# Customer Involvement

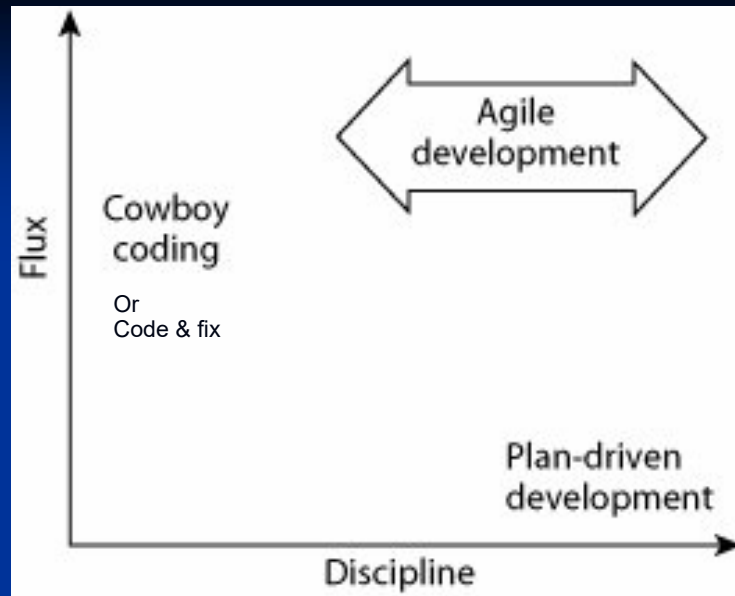


# The Agile Paradigm Shift









## Sample Productivity in Scrum

	Waterfall	Scrum
Size	3000 Use Case pages	1400 User Stories
Months	9	12
Person Months	540	54
Lines of Code in Java	58K	51K
Lines of code / Person-Month	120	840
Maximum team size	100	7

Reference: User Stories Applied – Mike Cohn