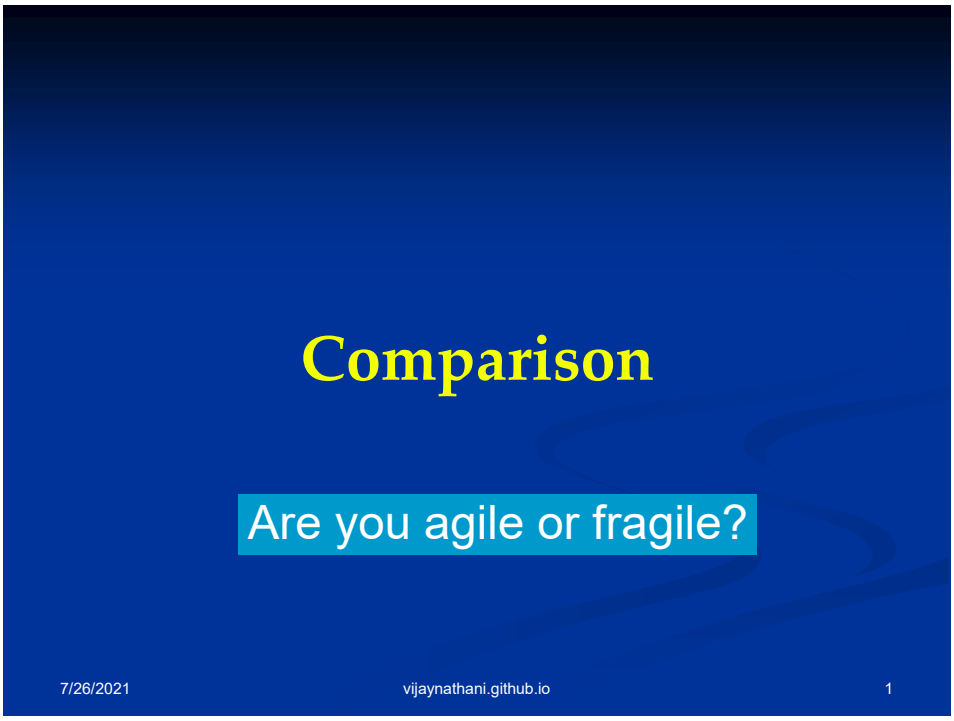


Swimming is a process even free-style swimming.

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*

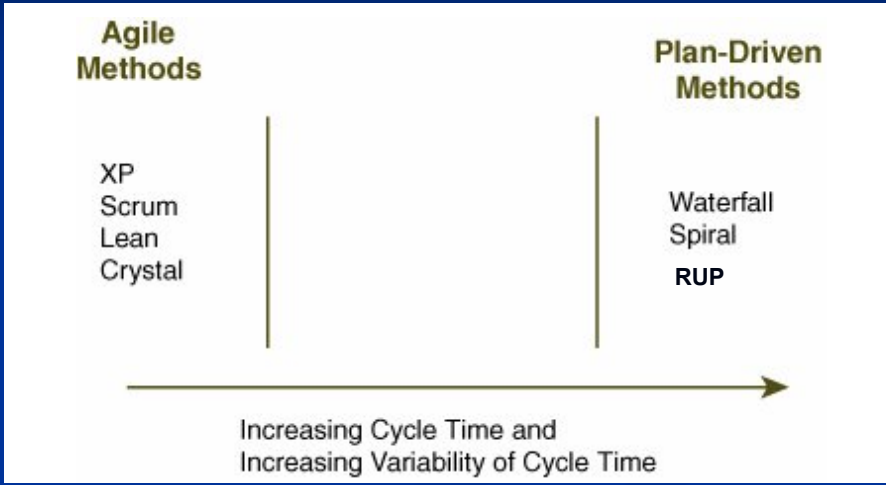
•pay up or

•write the project off and start again.

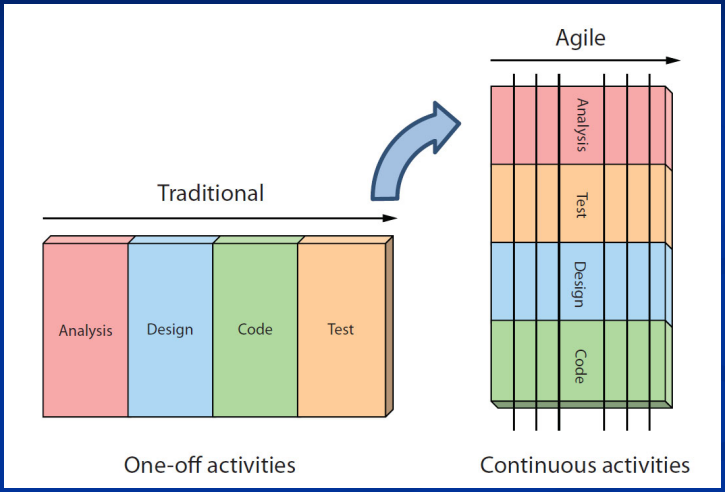
The client is often better off writing off their initial investment and starting over but its amazing how often they don't do that and continue to burn money.
Keith Sterling responded: *This is why so many large consultancies stick to the waterfall method. By bidding low and stipulating a waterfall approach, 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.

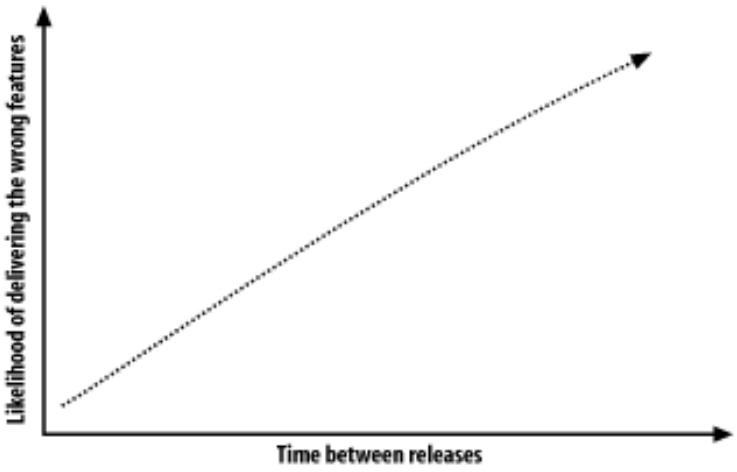
Comparison


Agile Methods
Plan-Driven Methods
XP
Scrum
Lean
Crystal
Waterfall
Spiral
RUP
Increasing Cycle Time and
Increasing Variability of Cycle Time

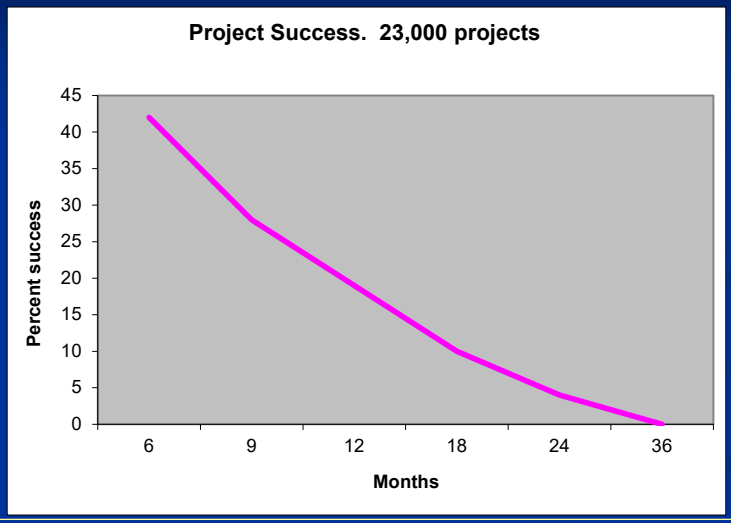
Activities


Agile
Traditional
Analysis
Design
Code
Test
Analysis
Test
Design
Code
One-off activities
Continuous activities

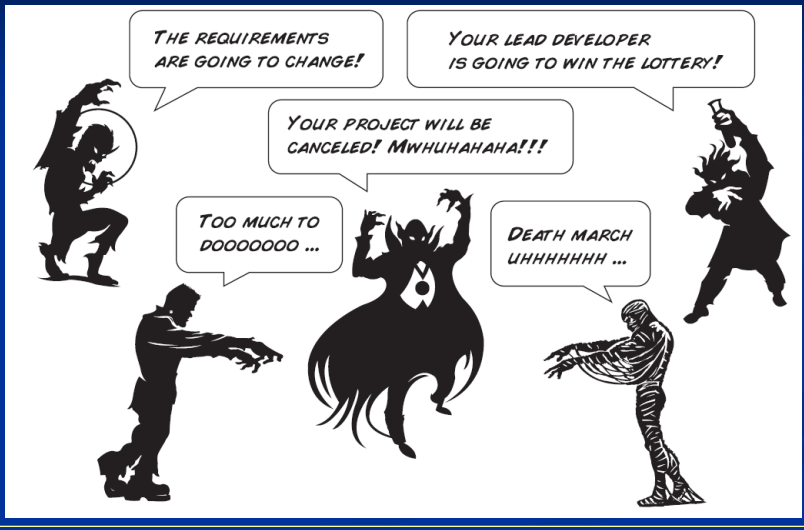
Comparison


Likelihood of delivering the wrong features
Time between releases

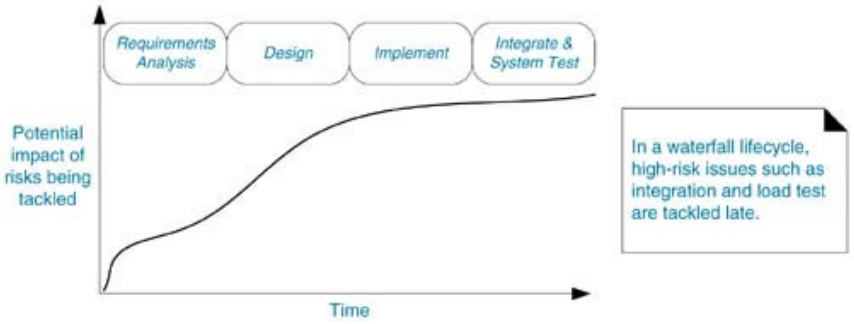
Long Projects Fail.



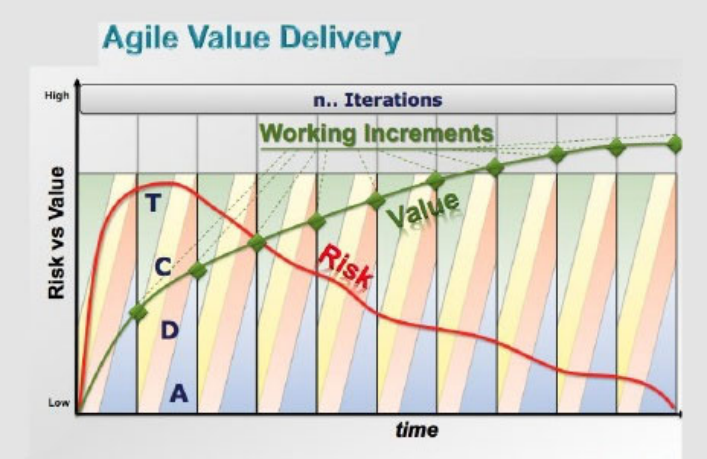
Risk

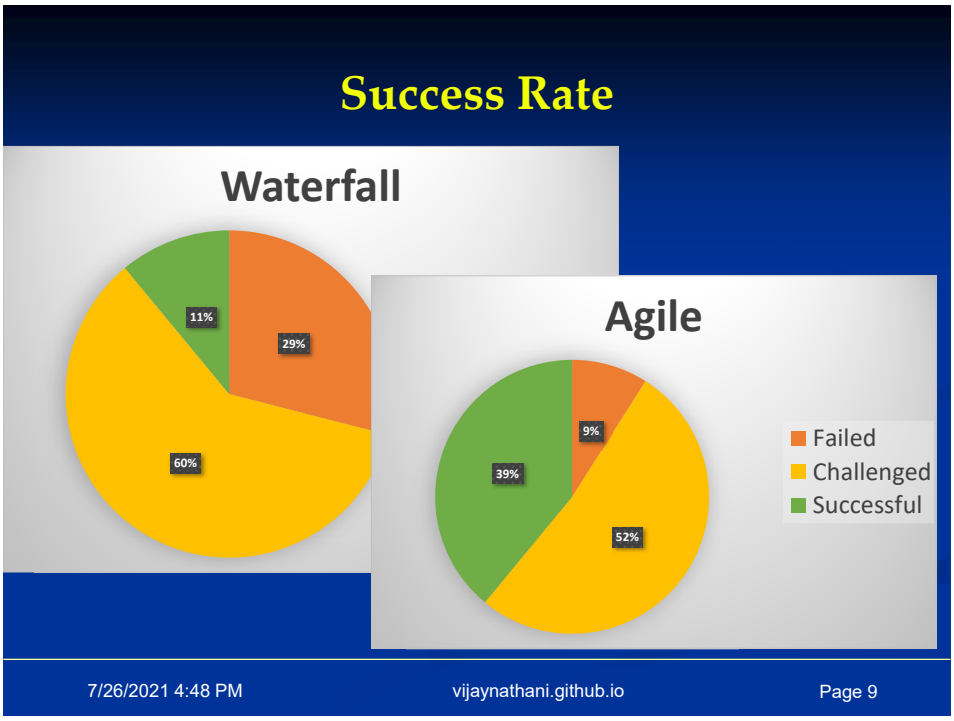

THE REQUIREMENTS ARE GOING TO CHANGE!
YOUR LEAD DEVELOPER IS GOING TO WIN THE LOTTERY!
YOUR PROJECT WILL BE CANCELED! MWHUHAHAHA!!!
TOO MUCH TO DOOOOOOO ...
DEATH MARCH UHHHHHHH ...

Risk in Waterfall



Risk / Value in Iterative

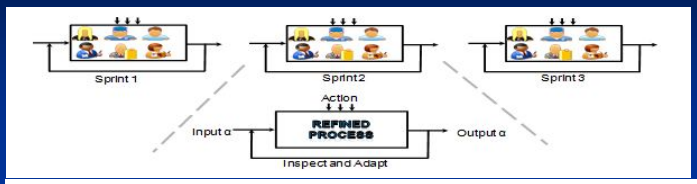


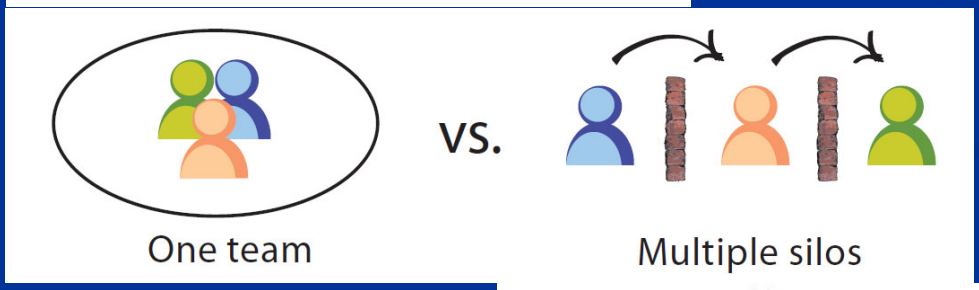


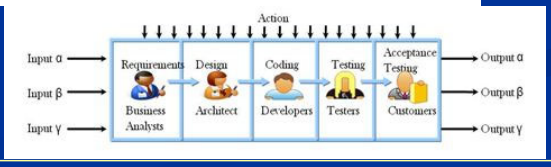
Standish Group Chaos Report 2015. Reference
http://www.infoq.com/articles/standish-chaos-2015

More details in AgileVsTraditional.png

Agile Team vs. Traditional Team



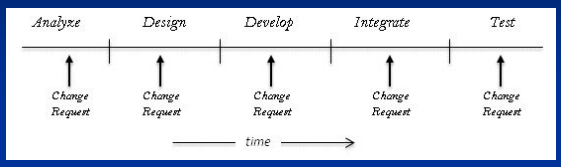


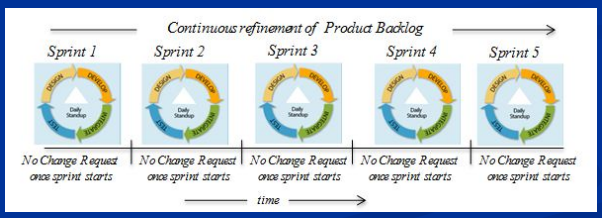


Silo – Pit dug for some specific purpose i.e. Sepration

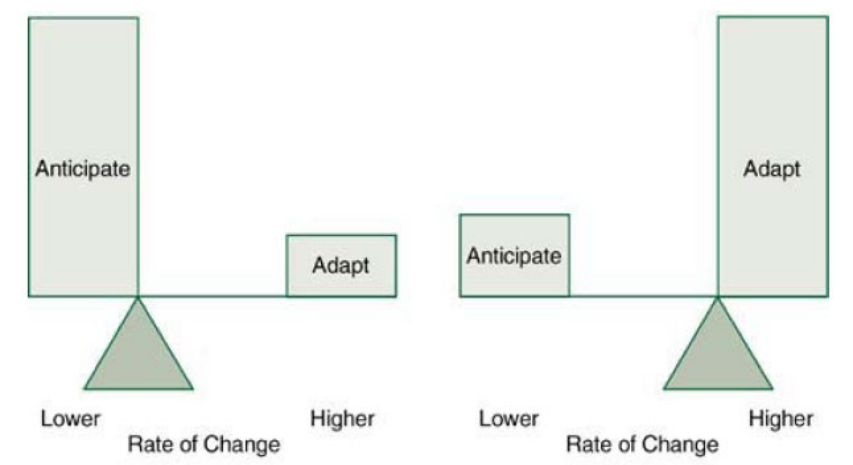
No seprate QA. Quality is everyone's responsibility.

Change Requests





Balancing



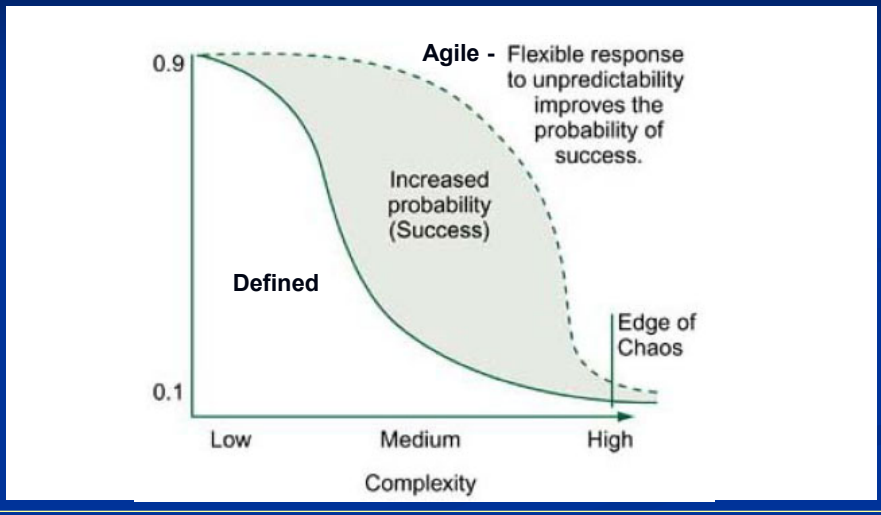
Three things we wish were true

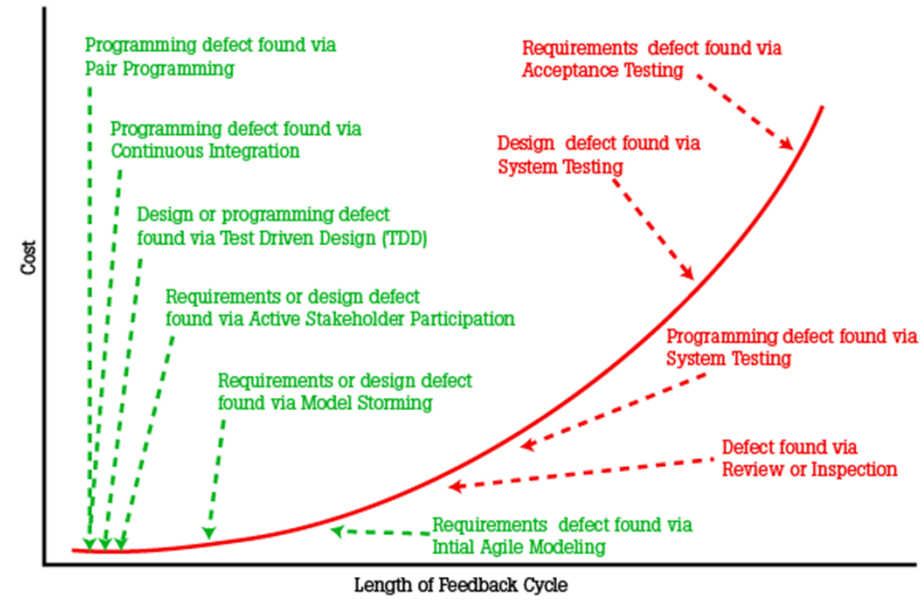
1. The customer knows what she wants
2. The developers knows how to build it
3. Nothing will change along the way

Three things we have to live with

1. The customer discovers what he wants
2. The developers discover how to build it
3. Lots of things change along the way

Flexible


0.9
0.1
Agile - Flexible response to unpredictability improves the probability of success.
Increased probability (Success)
Defined
Edge of Chaos
Low
Medium
High
Complexity



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.

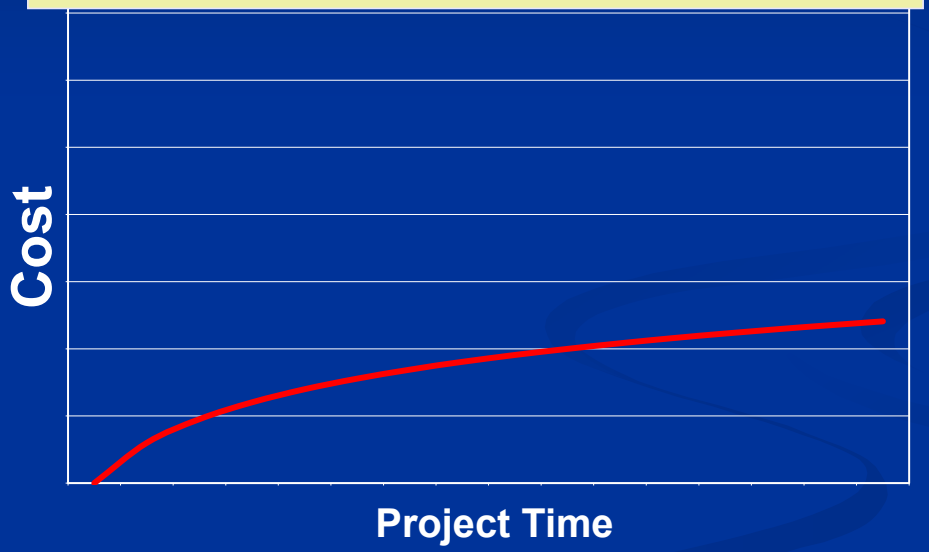
• Diminishing Returns. As more investment in something is made, the overall return on investment (ROI) increases at a declining rate until it reaches a point where it begins to decline. For example, if a diagram isn't yet good enough for the situation at hand, then you can keep working on it until it 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.

→ 1,000,000 X power/$!!

Compile/test cycle has gone from days to seconds.

We have CI and automated tests.

Compile cycles have gone done 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.

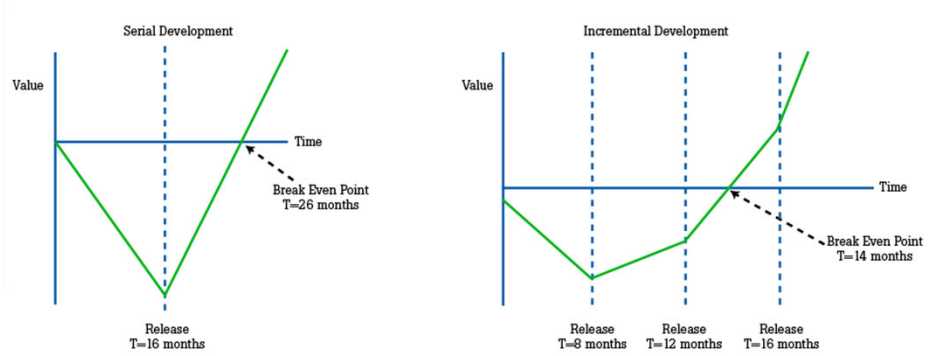
If you are uncertain and you can wait, then the risk goes away over time.

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



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.

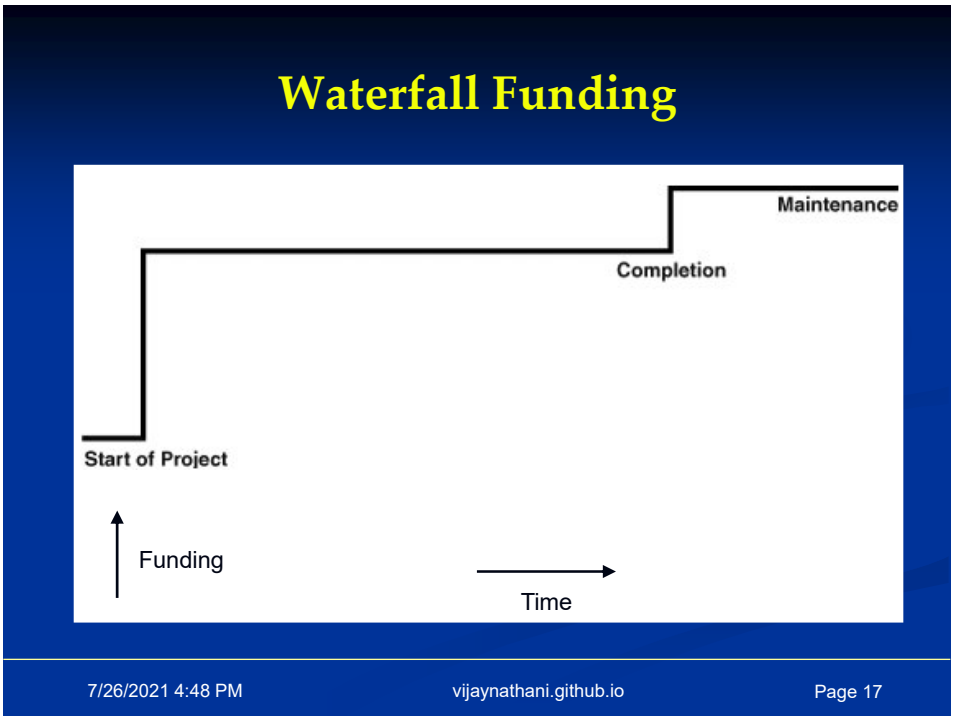
•Opportunity Cost. This is the cost of passing up a choice when making a decision. For example, if you could have spent $10,000 in additional testing and avoided a mistake that ended up costing you $15,000 to fix, then the opportunity cost of not testing was $5,000 ($15,000—$10,000). The concept of opportunity 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) x 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 x $50,000) and with a discount rate of 5% the TCO is $886,087.



Estimation is a black art.

> The typical software organization is not struggling to improve its estimates from ±10% to ±5% accuracy. The typical software organization is struggling to avoid estimates that are incorrect by 100% or more.

The more people who work on the project, the more total effort is usually required.

> A very rapid build-up of people often correlates with schedule slippage.

A Waterfall methodology project is usually on-time or ahead of schedule when it starts. It remains 90% complete for a very long time before and after the deadline.

Although there are many reasons for uncertainty, incomplete information of the problem requirements dominates.

Our techniques of estimating are poorly developed. More seriously, they reflect an unvoiced assumption that is quite untrue i.e. that all go well…Because we are uncertain of our estimates, software mangers often lack the courteous stubbornness to make people wait for a good product.

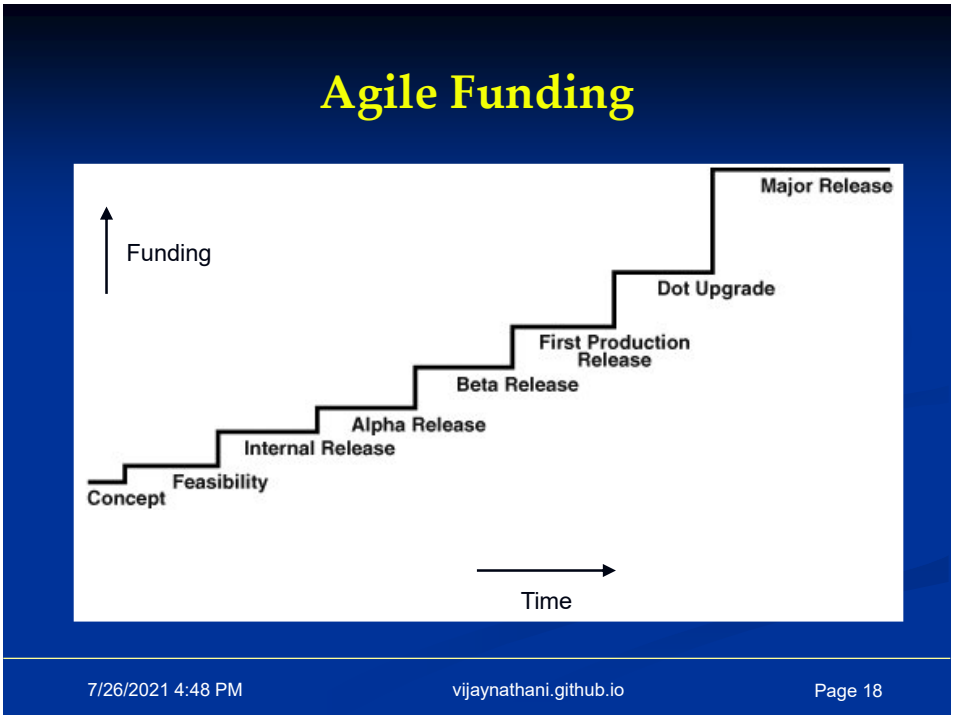
Be Honest in Estimates:

Estimate is an approximate judgment or opinion of the value of a measure, but many managers use the same term to mean a promise by the developers.

Be honest, and have the courage to communicate the truth. It may be difficult at

times; that's why it takes courage.

> Most people suffer, not because of external circumstances, but because of lack of courage.
>
> Napoleon once said, "Any commander-in-chief who undertakes to carry out a plan which he considers defective is at fault; he must put forth his reasons, insist on the plan being changed, and finally tender his resignation rather than be the instrument of his army's downfall." These are strong words that many software project managers should ponder.



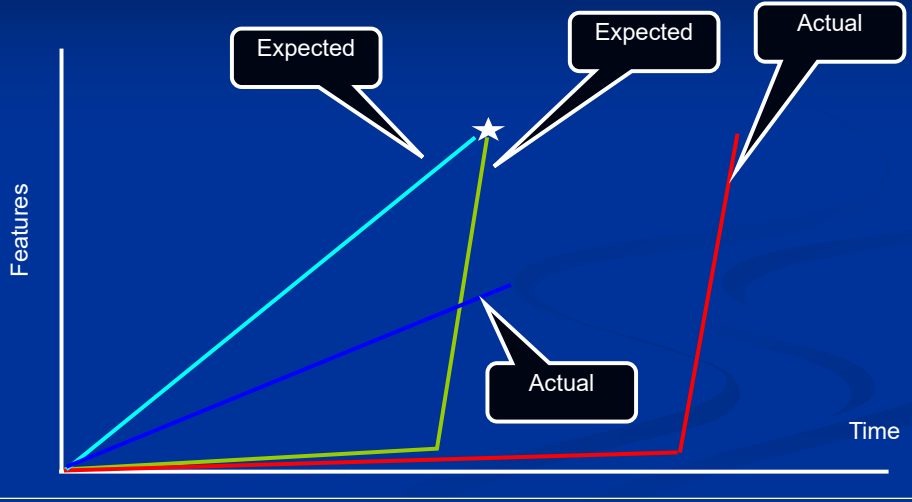
"Fixed price". At the beginning of the project develop, and then commit to, an initial estimate based on your up-front requirements and architecture modeling efforts. Hopefully this estimate is given as a range, studies have shown that up-front estimating techniques such as COCOMO II or function points are accurate within +/- 30% most of the time although my July 2009 State of the IT Union survey found that on average organizations are shooting for +/- 11% (their actuals come in at +/- 19% on average, but only after doing things such as dropping scope, changing the estimate, or changing the schedule). Fixed-price funding strategies are very risky in practice because they promote poor behavior on the part of development teams to overcome the risks foisted upon them as the result of this poor business decision. It is possible to do agile on a fixed budget but I really wouldn't recommend it (nor would I recommend it for traditional projects). If you're forced to take a fixed-price approach, and many teams are because the business hopes to reduce their financial risk via this approach not realizing that it actually increases their risk, then adopt strategies that reduce the risk.

Stage gate. Estimate and then fund the project for given periods of time. For example, fund the project for a 3-month period then evaluate it's viability, providing funding for another period of time only to the extent that it makes sense. Note that stages don't have to be based on specific time periods, they could instead be based on goals such as to initiate the project, prove the architecture with working code, or to build a portion of the system. Disciplined agile methods such as Open Unified Process have built in stage-gate decision points which enable this

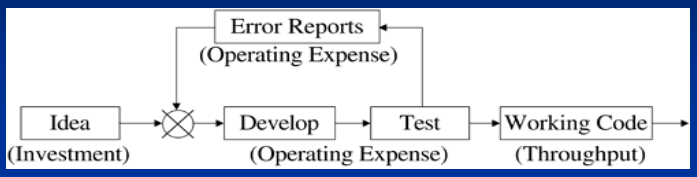
sort of strategy. When the estimation technique is pragmatic, the best approaches are to have either the team itself provide an estimate for the next stage or to have an expert provide a good gut feel estimate (or better yet have the expert work with the team to develop the estimate). Complex approaches such as COCOMO II or SLIM are often little more than a process facade covering up the fact that software estimating is more of an art or a science, and prove to be costly and time consuming in practice.

Time and materials (T&M). With this approach you pay as you go, requiring your management team to actually govern the project effectively. Many organizations believe a T&M strategy to be very risky, which it is when your IT governance strategy isn't very effective. An interesting variation, particularly in a situation where a service provider is doing the development, is an approach where a low rate is paid for their time which covers their basic costs, the cost of materials is paid out directly, and delivery bonuses are paid for working software. This spreads the risk between the customer/stakeholder and the service provider. The service provider has their costs covered but won't make a profit unless they consistently deliver quality software.

Project Execution



How Organizations Become More Responsive



*D. Anderson: *Agile Software Management Accounting for Systems*

Three Levers to Increase Responsiveness	
Increase Throughput	Shorten cycle times to speed delivery of functionality to customers
Decrease Investment	Reduce inventory and the cost capture, elaboration, communication and scheduling
Decrease Operating Expense	Eliminate waste to produce higher quality with less effort

Agile can change the economic model by a factor of four. - D. Anderson: *Agile Software Management Accounting for Systems.*

Net Profit = Throughput – Operating Expense

ROI = Net Profit / Investment

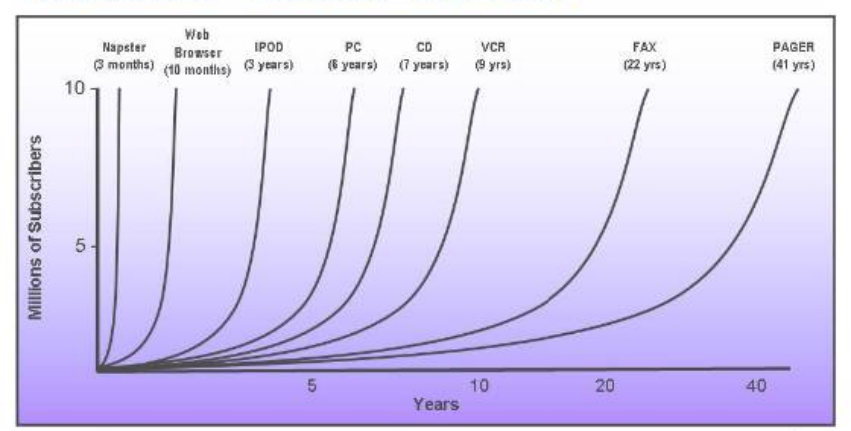
Agile processes are characterized by sustainability and responsiveness:

•Feedback at various levels of scale, weather in terms of customer goals, test results, daily team feedback, etc.

•A balance between shortening time to value and reducing accumulation of technical debt.

•Agile development puts people back in the picture, instead of the process.

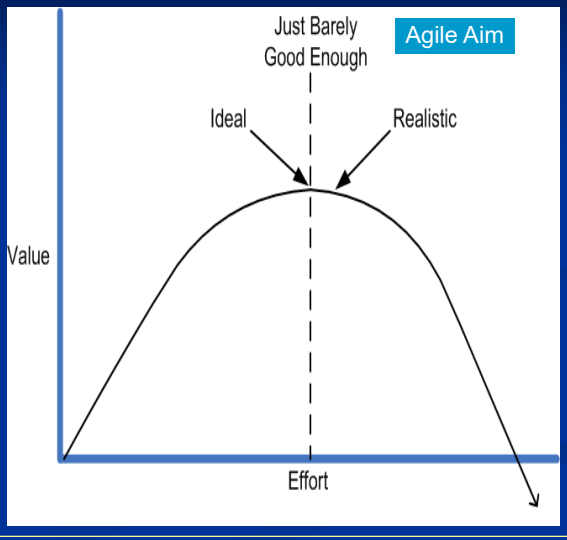
Competitiveness – Accelerating Pace of change



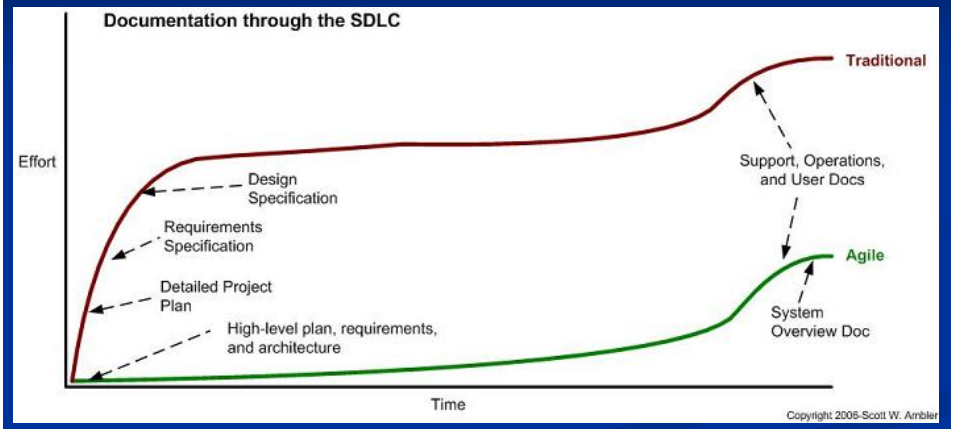
For most companies: Time to market is more important than the cost

Documentation

Traditional



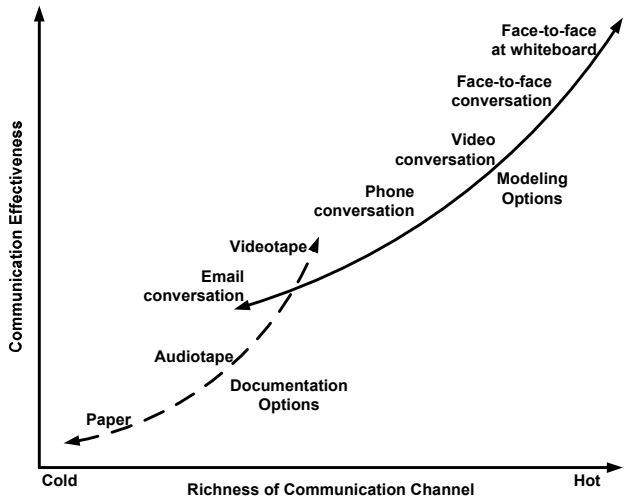
Comparison of Documentation



In Agile, documentation is typically 5% of waterfall.

Communication Modes

Always Strive to Use the Most Effective Approach



Software development is a communication game

Documentation is the worst way to communicate

Whatever your situation, use the most effective means at your disposal to communicate

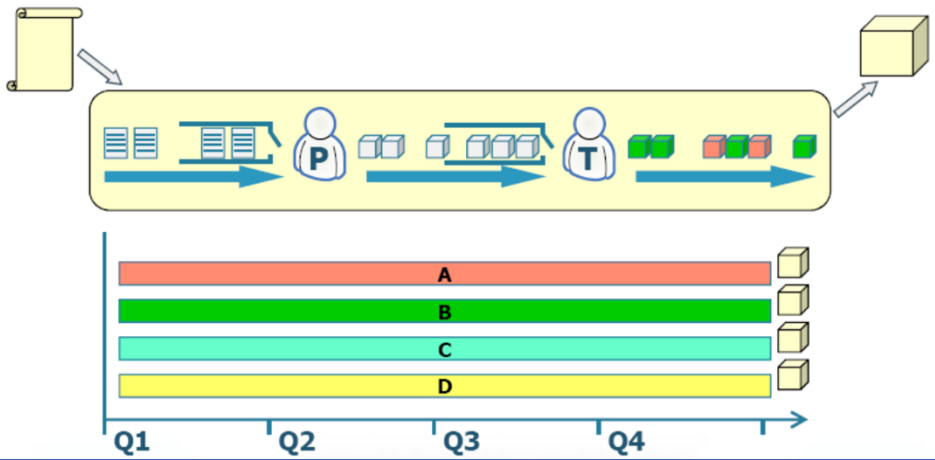
http://www.agilemodeling.com/essays/communication.htm

One big Restaurant on its menu card said “Main dish comes with soup or salad and bread”.

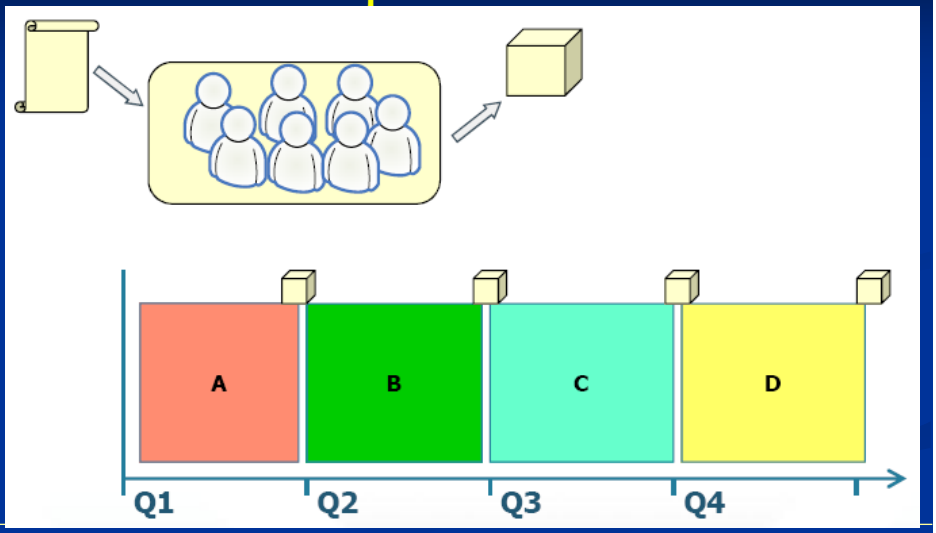
Words are imprecise e.g. Does this mean “(soup) or (salad and bread)” or does it mean “(soup or salad) and bread”. I was confused. The only way to clarify was to ask the waiter. When I called the waiter, he brought bread, kept in on the table and asked “Soup or Salad?”.

Now the meaning is clear.

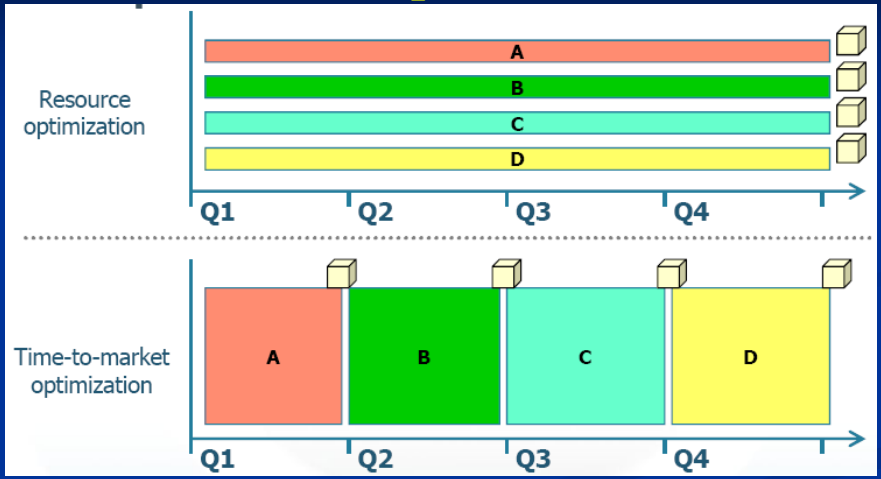
Traditional Company: Resource Optimization



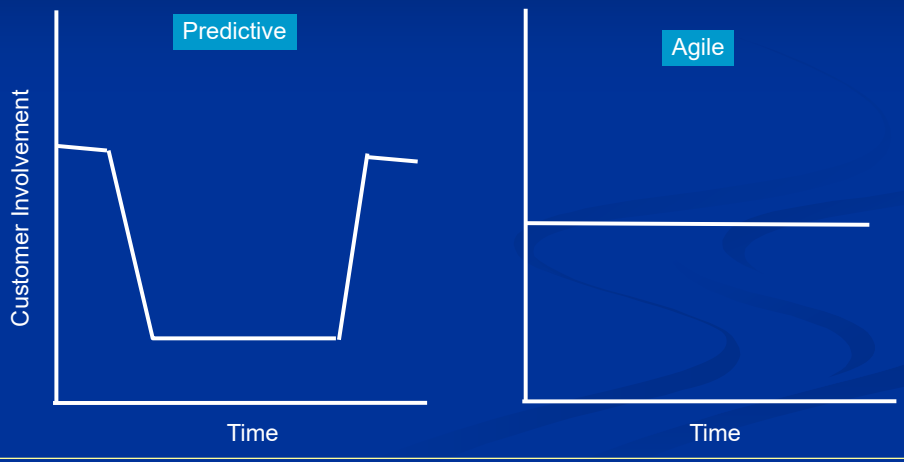
Agile Company: Time to Market Optimization

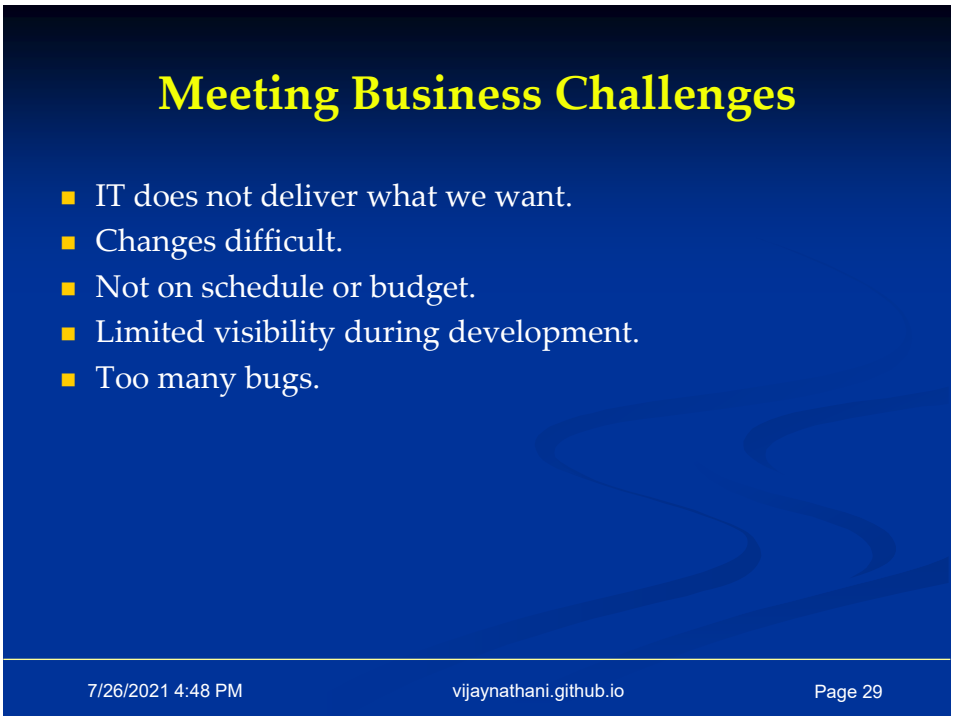


Comparison


Resource optimization
A
B
C
D
Q1
Q2
Q3
Q4
Time-to-market optimization
A
B
C
D
Q1
Q2
Q3
Q4

Customer Involvement





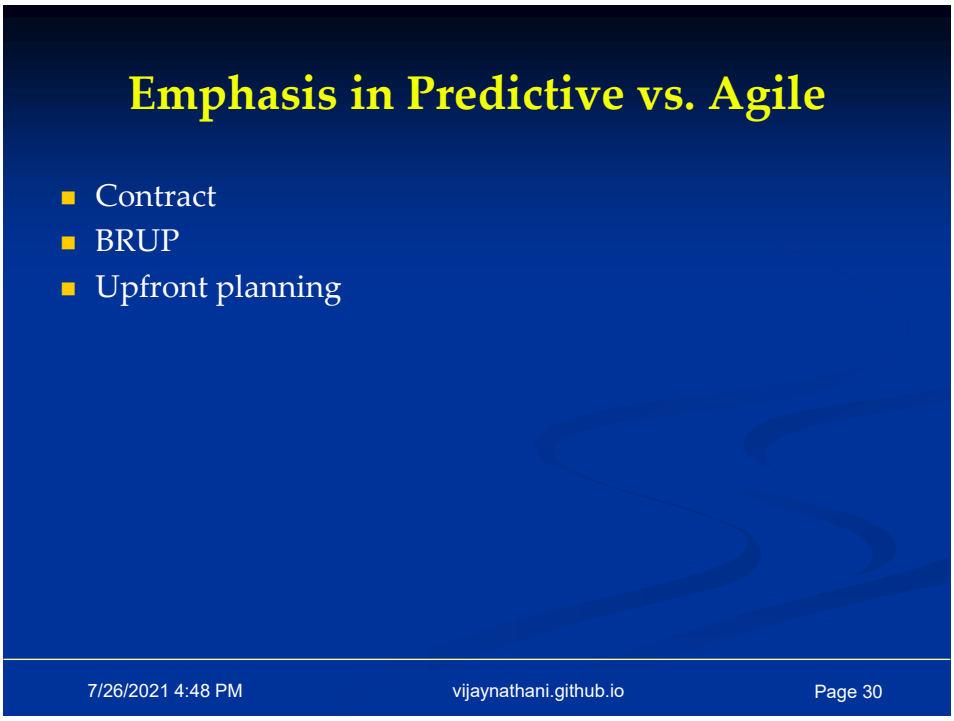
Business Challenge: IT does not deliver what we want. **IT Challenge:** Business does not know what it wants. **Agile approach:** Business and IT collaborate throughout the project.

Business Challenge: Process not flexible enough to allow scope changes. **IT Challenge:** Requirements and scope change too often. **Agile approach:** Process allows and minimizes the impact of change.

Business Challenge: Deliver not on schedule or budget. **IT Challenge:** Ambitious delivery deadlines, limited budget. **Agile approach:** Fixed price budget achieved by delivering prioritized functionality.

Business Challenge & IT Challenge: Limited visibility to monitor weekly progress. **Agile approach:** Working functionality delivered every 2 weeks

Business Challenge: System has too many bugs. **IT Challenge:** Not enough time to fully test the system. **Agile approach:** Continuous testing resulting in zero defects.



Traditional: Focus on agreeing to a detailed contract with customers about the totality of the system to be delivered along with the costs and time scales.

Agile: Focus on setting up a collaborative relationship with the customers.

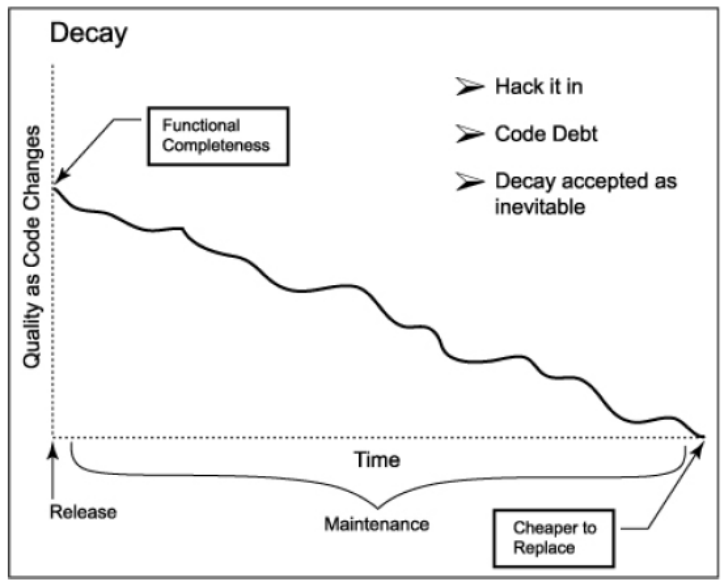
Traditional: We are concerned with understanding the requirements in complete detail

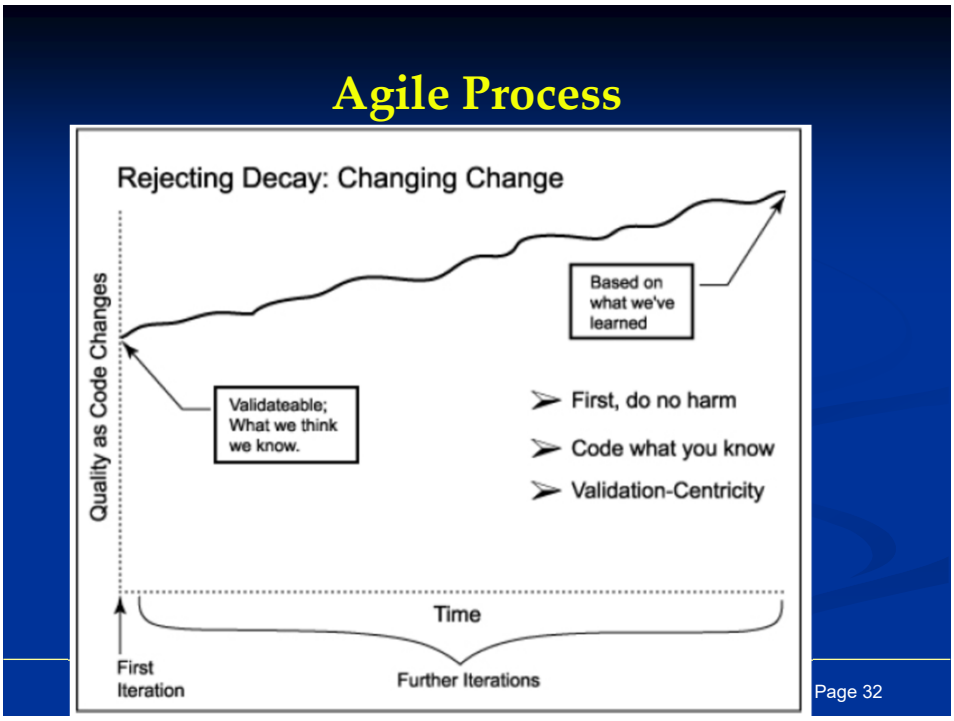
Agile: We are concerned with agreeing with the users on the process by which the business requirements will be met.

Traditional: The plan is created in great detail and is ideally executed with minimal change

Agile: The initial plans are created in sufficient detail to establish the main parameters of the project with the firm expectation that the customers will change the plan during the course of the project

Planned Process





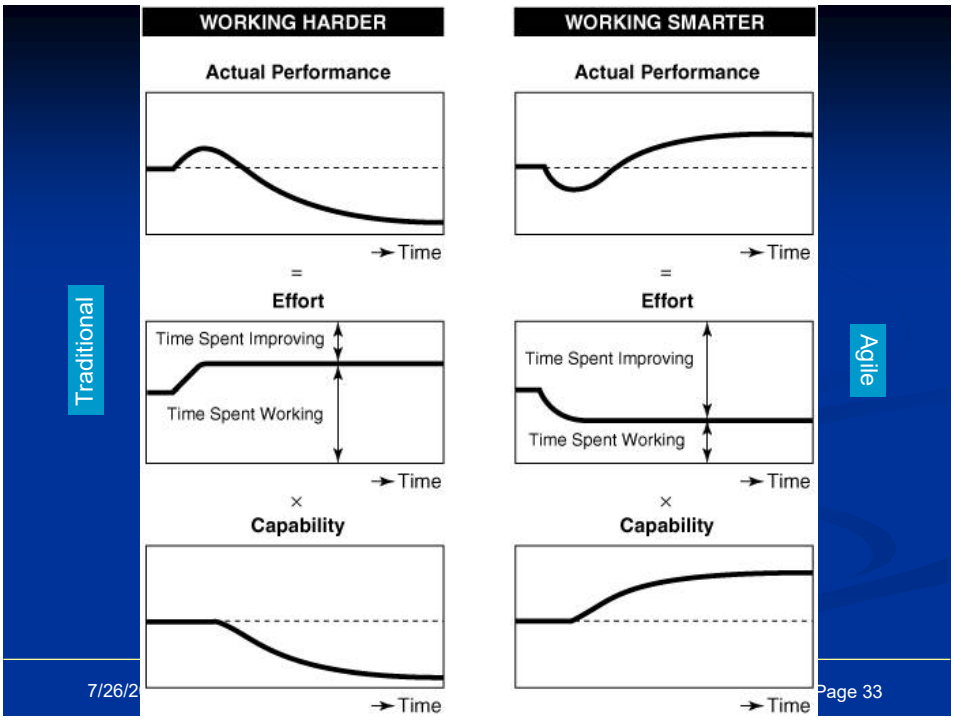
I say that decay is not inevitable if we refuse to accept it.

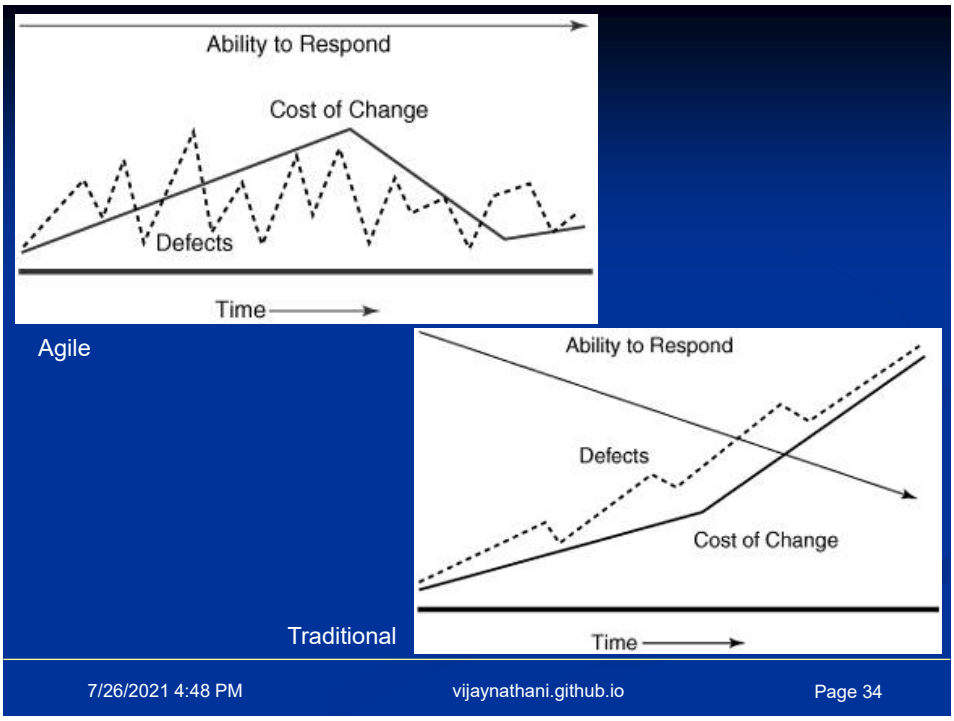
For this to be true, we must assert several things.

• We need something like the Hippocratic Oath: ”First, do no harm.” I do not expect my software to be perfect (ever), but I think it is reasonable to hold myself to the basic standard that every time I touch it, I will take care not to make it any worse.

• We need to code in a style that allows us to follow this do no harm oath.

•We need to center ourselves on the notion that validating software (with those who will use it) is part of making the software.


Traditional
WORKING HARDER
Actual Performance
Time
=
Effort
Time Spent Improving
Time Spent Working
Time
×
Capability
Time
WORKING SMARTER
Actual Performance
Time
=
Effort
Time Spent Improving
Time Spent Working
Time
×
Capability
Time
Agile



A predictable organization does not guess about the future and call it a plan; it develops the capacity to rapidly respond to the future as it unfolds.

—Mary Poppendieck

Heavyweight vs. Lightweight

Predictive: well defined process, documents drive development, stable requirements

Agile: agile process, development = testing + coding, changing requirements

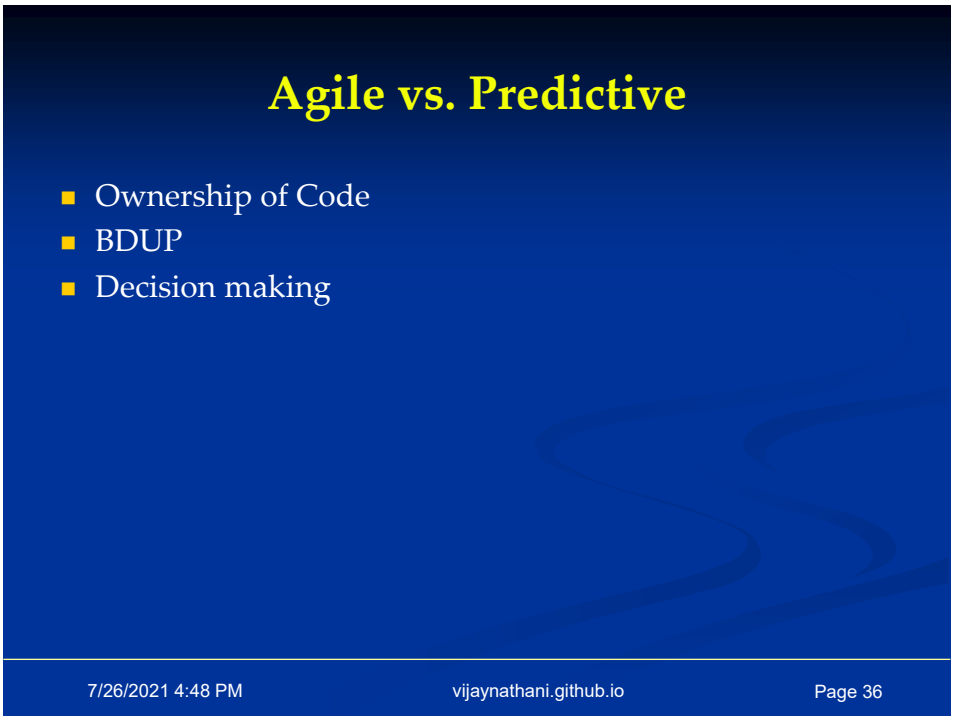
Predictive: BRUP. Agile: Learning driven. Details of requirements defined just-in-time

Predictive: Deliver in 'big-bang'. Typically after a couple of months. Agile: 1 to 3 weeks iteration time.

Predictive: View programming as construction. Agile: View programming as design

Predictive: Integration and Testing usually at the end of a phase. Agile: Continuous integration. Fully-automated continuous testing (Unit and functional).

Predictive: High cost of change. Resist change. Agile: Low cost of change. Welcome change.



Traditional: Behavior is predictable and controllable. Agile: Behavior is unpredictable and uncontrollable

Traditional: Direction is determined by a few leaders. Agile: Direction is determined though emergence and by many people

Traditional: Every effect has a cause. Agile: Every effect is also a cause

Traditional: Relationships are directive. Agile: Relationships are empowering

Traditional: Efficiency and Reliability are measures of value. Agile: Responsiveness to environment is the measure of value

Traditional: Decisions are based on facts and data. Agile: Decisions are based on patterns and tensions

Traditional: Leaders are experts and authorities. Agile: Leaders are facilitators and supporters.

Predictive	**Agile**
Subsystem ownership	Collective ownership
Single design. BDUP (Big design up-front).	Incremental development
Comprehensive documentation	Automated test. Non-software artifacts to approach zero.
Quality by analyzing complete design	Quality by TDD (Test driven development) and Automated Functional tests.
Individual design decision.	Team consensus
Command &Control management	Self-managed

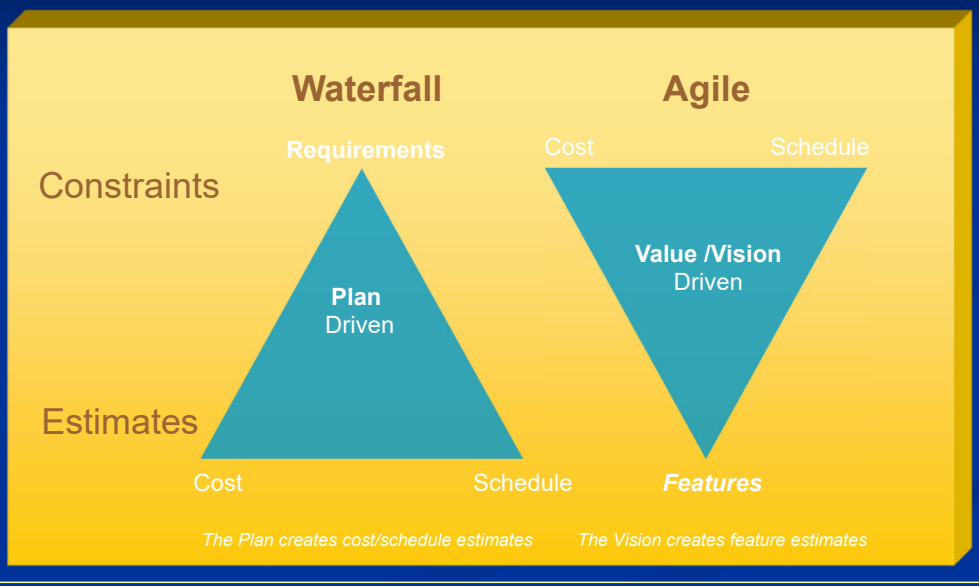
Strict Change management	Change is inevitable
Comprehensive documentation	Working software. Get feedback by face-to-face communication.
Up-front planning	Plan to the next iteration
Formal entrance and exit criteria with signoffs.	Collaborate, don’t sign off
Comprehensive system-level regression tests.	In addition to system-level regression test, use TDD and CI for detecting defects earlier

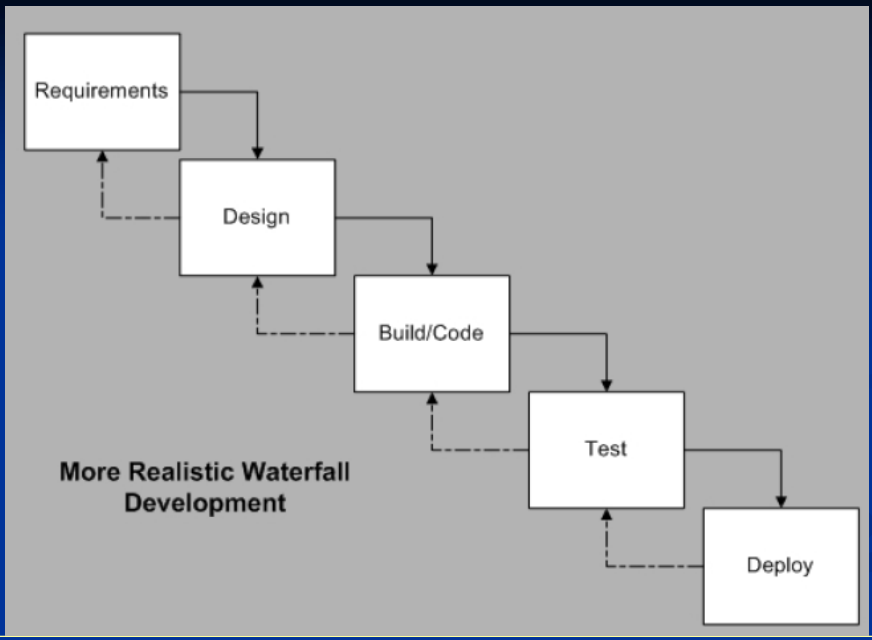
What Paradigms are changing?

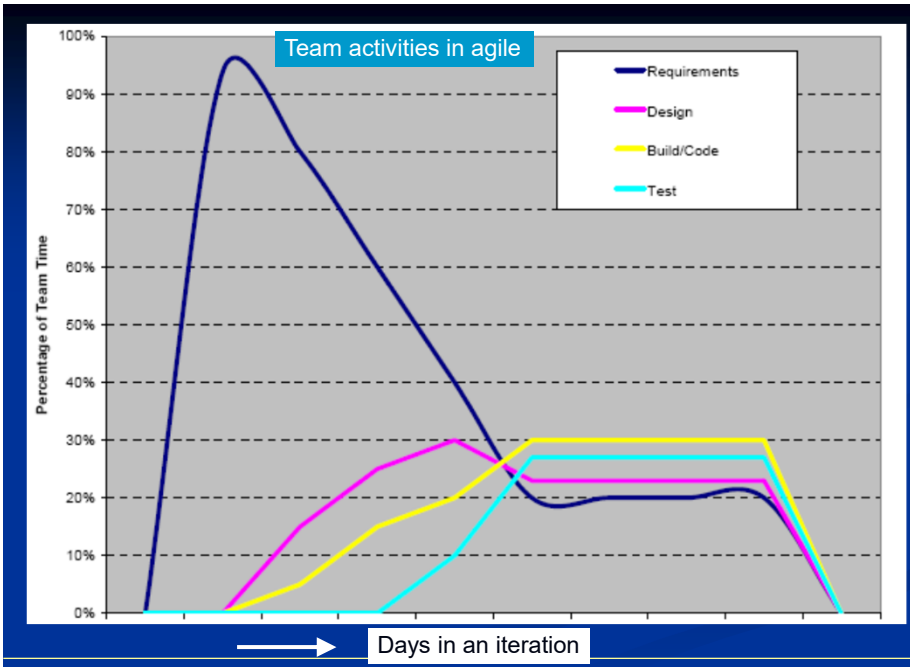
- Measure of Success
- Culture
- Design
- QA
- Inventory
- Feedback

Waterfall	**Agile**
Conformance to plan	Customer Satisfaction
Command & Control	Collaborative
BDUF	Continuous
Big Test at end	Continuous
Inventory Huge	Minimal
Feedback Slow	Rapid

The Agile Paradigm Shift




Requirements
Design
Build/Code
Test
Deploy
More Realistic Waterfall Development


Team activities in agile
Requirements
Design
Build/Code
Test
Percentage of Team Time
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Days in an iteration

Myths

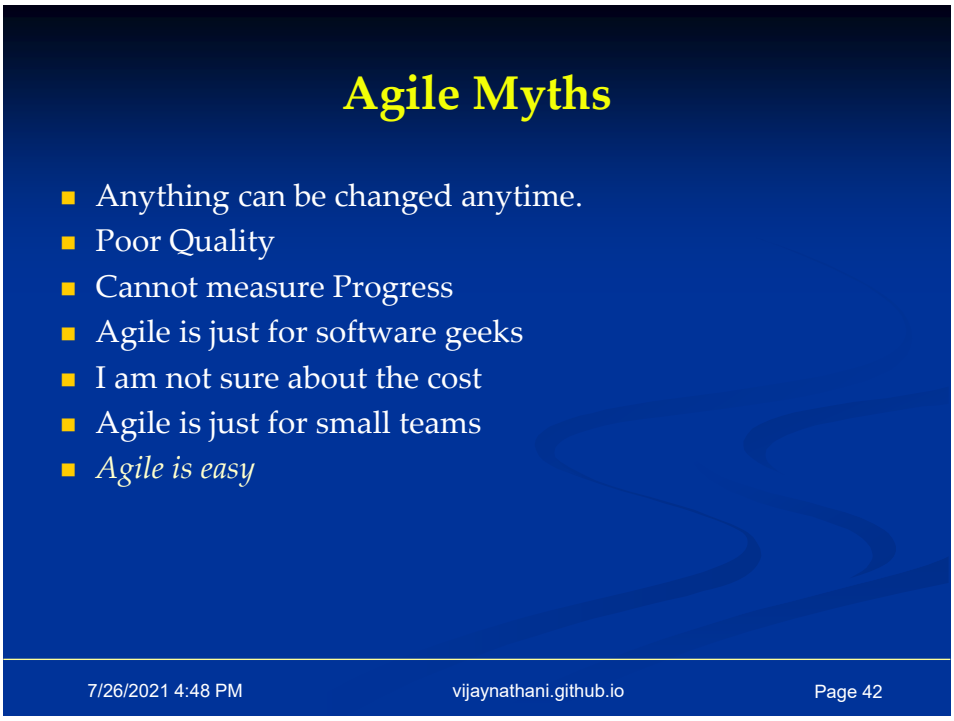
- Early specification reduces waste
- The job of testing is to find defects
- Predictions create predictability
- Software should not have to be changed once it is written.

Fact: YAGNI

Fact: Do we test the car after manufacturing to see if it runs?

Fact: Coding alone gives us enough feedback about the work

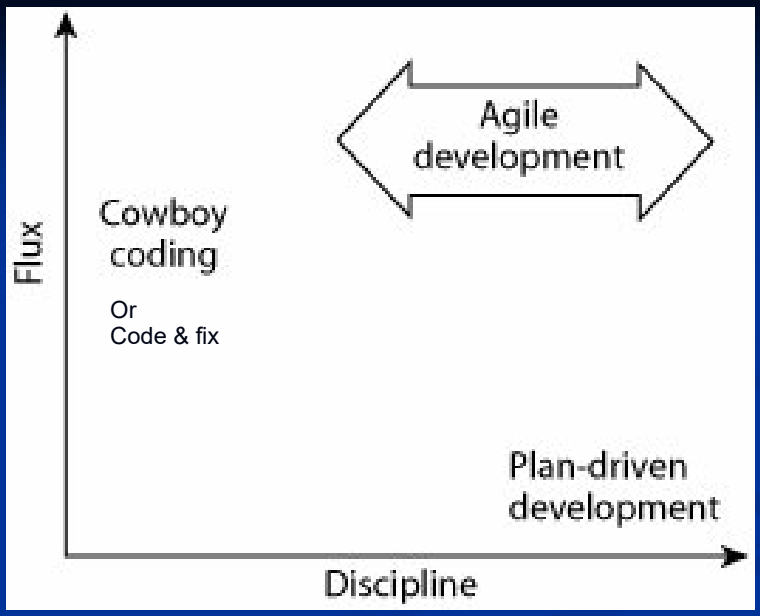
Fact: TDD and CI (no BDUP / BRUP) ensure that software can be changed easily

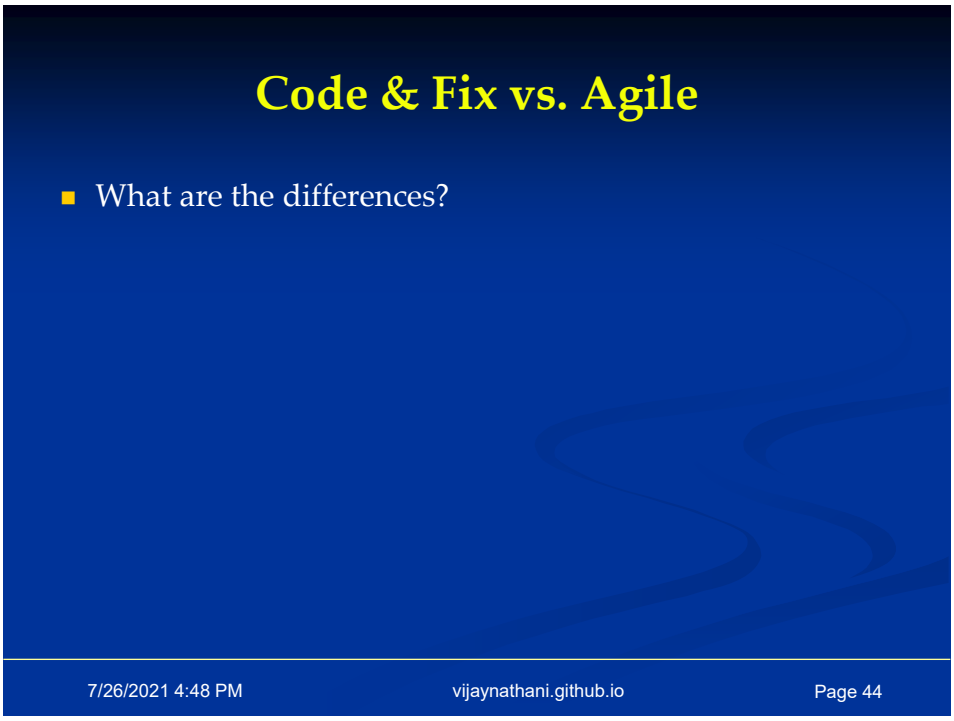


Myths are prominent:

•In a company, manager tells the employee “Come at 8. If you cannot come at 8, then come directly at lunch time”

•Cutting of Ham on both sides before cooking. Earlier pans were small, now they are large enough to hold the entire ham.


Agile
development
Cowboy
coding
Or
Code & fix
Flux
Plan-driven
development
Discipline



Code& Fix:

Test Coverage < 20%

Unable to deliver production grade code every two weeks, consistently over long periods

A few people tell others in the team the work to be done. Project manager or team lead will take decision on issues.

Haphazard development process, but usually call themselves agile.

Motivation level among team usually poor.

Lots of bugs / issues reported by the customer.

No pair-programming most of the time.

Testers not present or primarily responsible for finding defects.

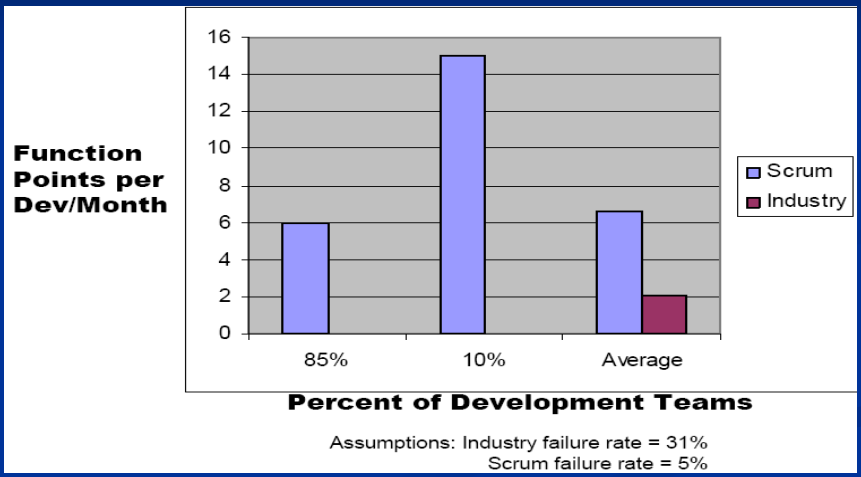
Higher scope at the cost of quality initially. High productivity at start, low productivity after a few months.

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

Improvement in Scrum



Assumptions: Industry failure rate = 31%
Scrum failure rate = 5%

Yahoo results

- Moved 50 projects with about 600 people to Scrum in 18 months. The results surveyed

Task	Better	No change	Worse
Productivity as per Team	68%	27%	5%
Team Morale	52%	39%	9%
Adaptability	63%	33%	4%
Accountability	62%	32%	6%
Collaboration & Co-operation	81%	18%	1%

Team productivity increased an average a 36% increase, based on the estimates of the Product Owners. 85% of team-members stated that they would continue using Scrum if the decision were solely up to them.