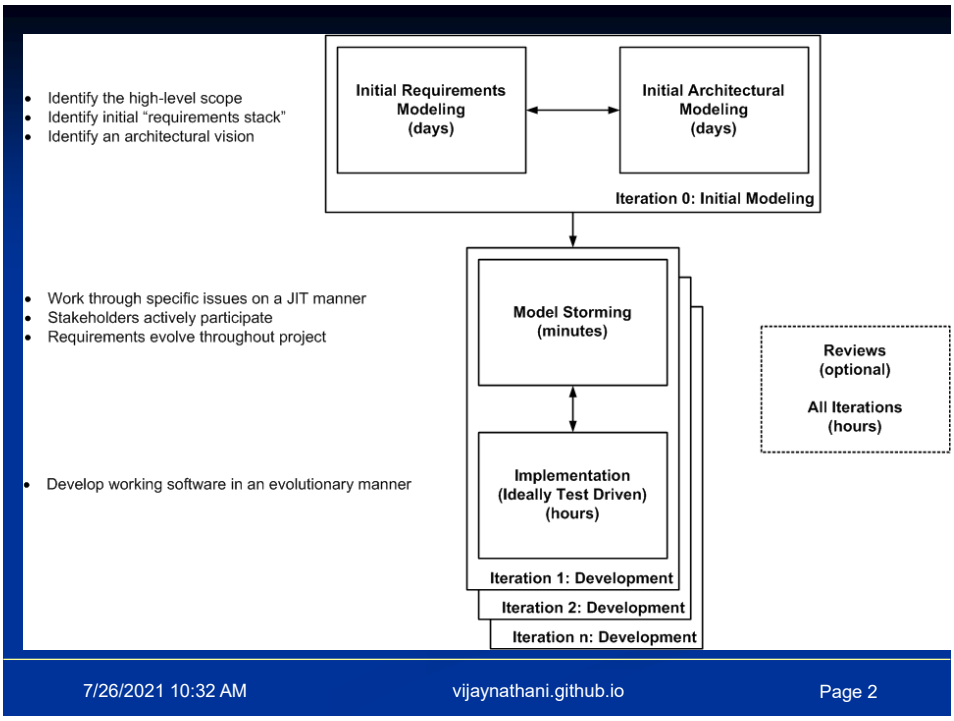
Ebay has 1000's of servers. It does a production release every two weeks.



What is Agile?

"Agility is the ability to both create and respond to change in order to profit in a turbulent business environment."—*Jim Highsmith, Agile Software Development Ecosystems*

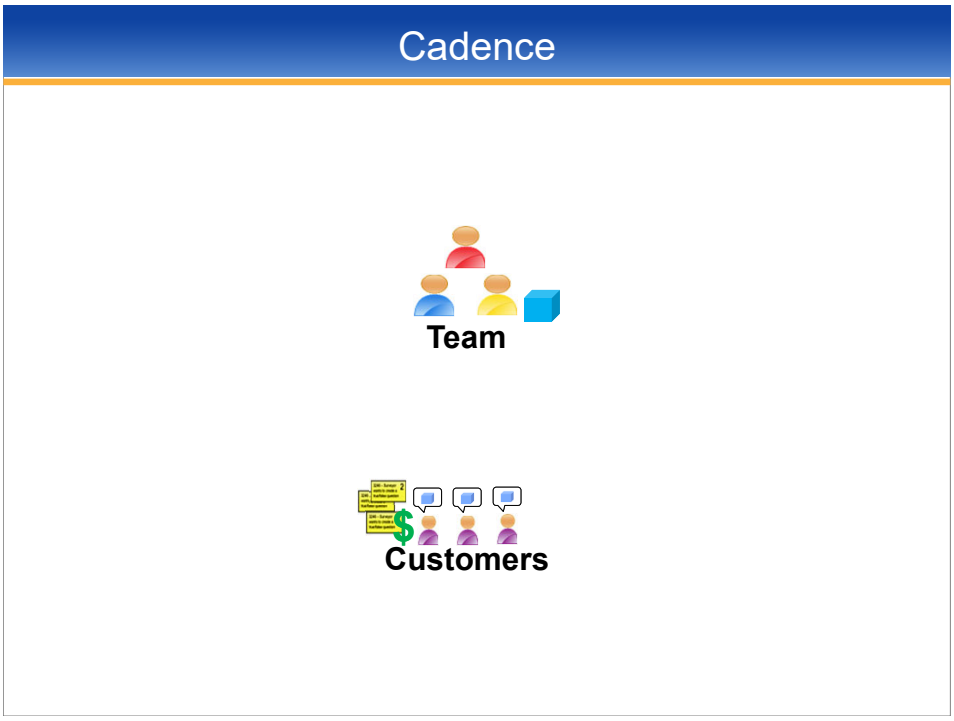
Nimbleness and flexibility balanced with structure

==========

Some architectural decisions need to be taken at the start e.g.

1. File or Database
2. Scalability of 100 or 10 million users
3. Database size less than 10GB or thousands of GBs
4. Java or .NET or Ruby

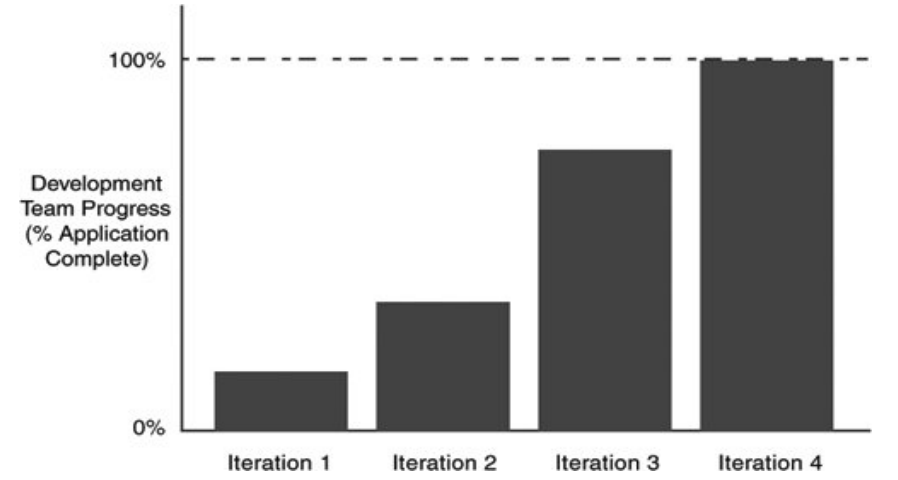
A decision that is reversible need not be taken at the start. An decision that is difficult to change later is an architectural decision.



Do as little as possible with the highest possible value as fast as possible.

Cadence means Rhythm.

Progress



Iteration refers to the process

•Cyclic process model is based on development periods of same duration.

•Duration is typical one to four weeks. The smaller the better.

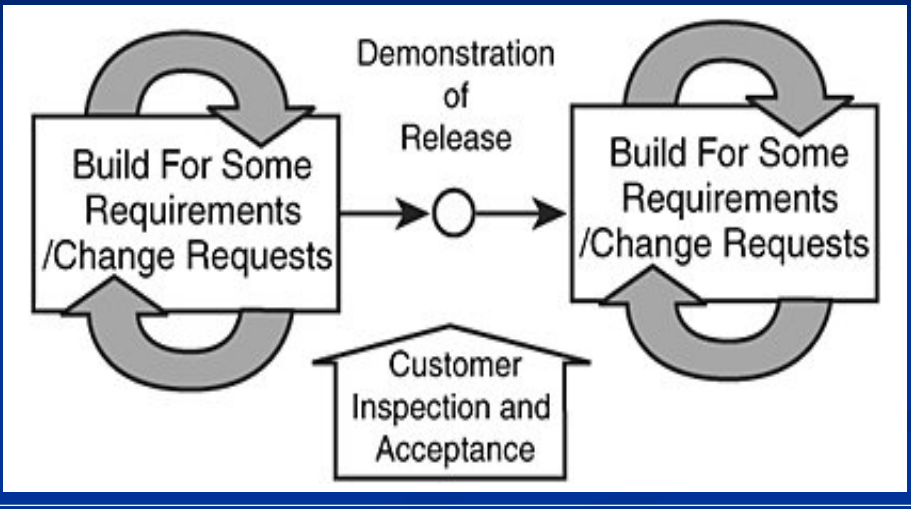
Iterations are time-boxed and have ritual-enforced boundaries. Progress feedback based on increments become meaningless, if the iteration duration is variable on demand. Everyone knows and feels when an iteration ends. It is signaled with a demo and a retrospective.

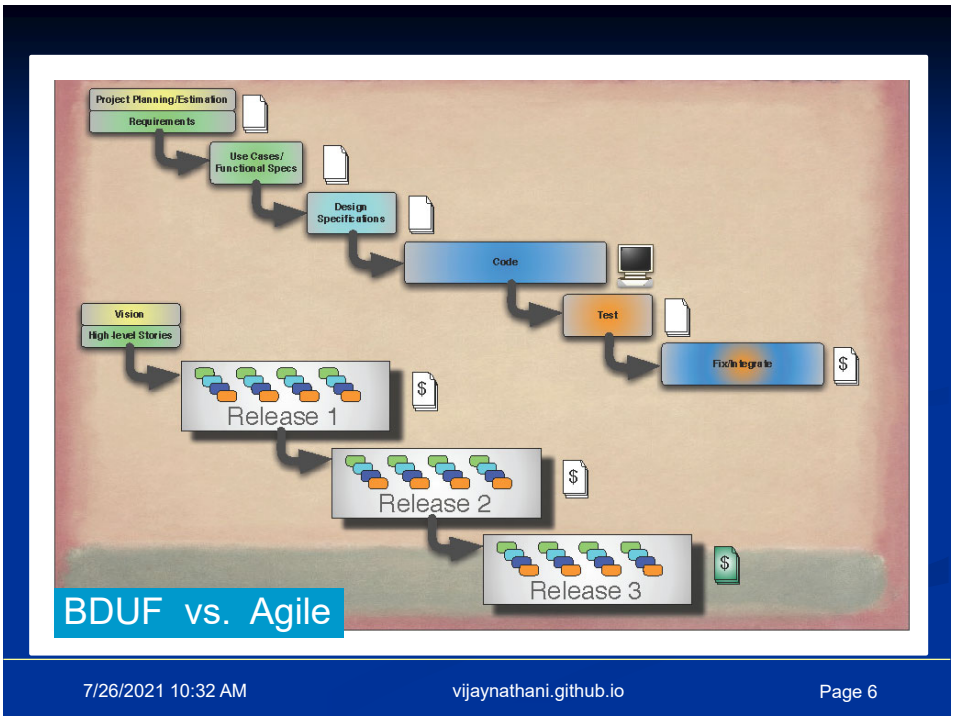
Increment refers to the product

•How much more product has been produced since the last iteration?

•Increment needs to be consumable rather than a software-centric artifact

Regular Customer Feedback



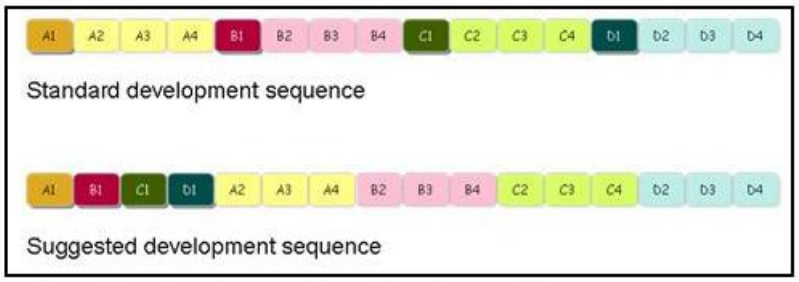


De-emphasizing dependencies between features

Time boxed more than Phased

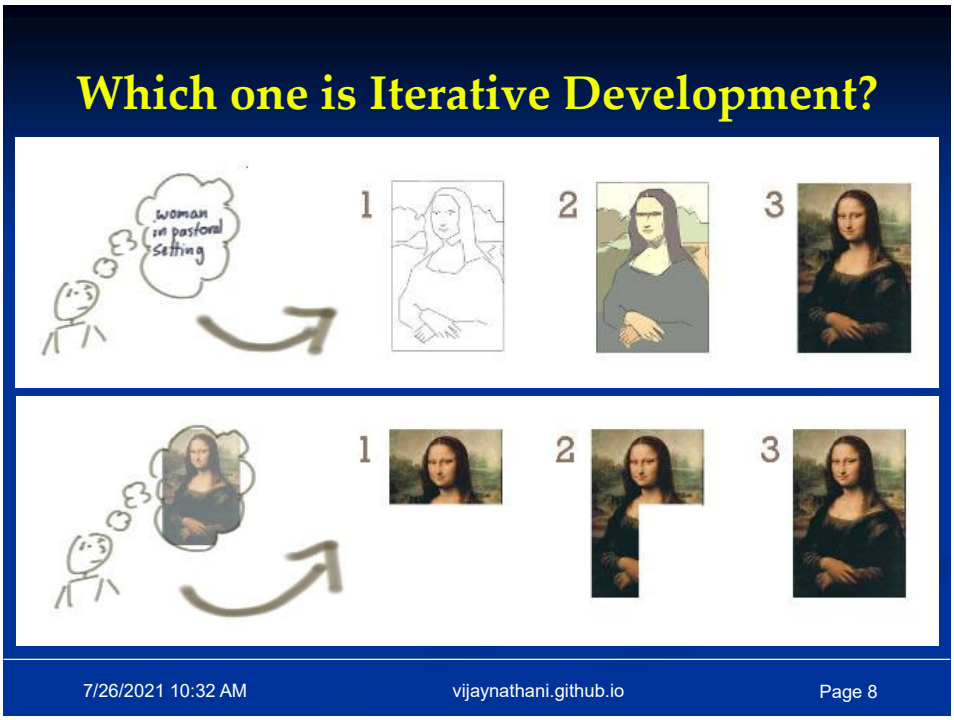
- Self-similar through the lifecycle
- Measured project velocity
- Continuous specifications, design, coding and testing activities.
- Early development validates planning estimates

Predictive vs. Agile

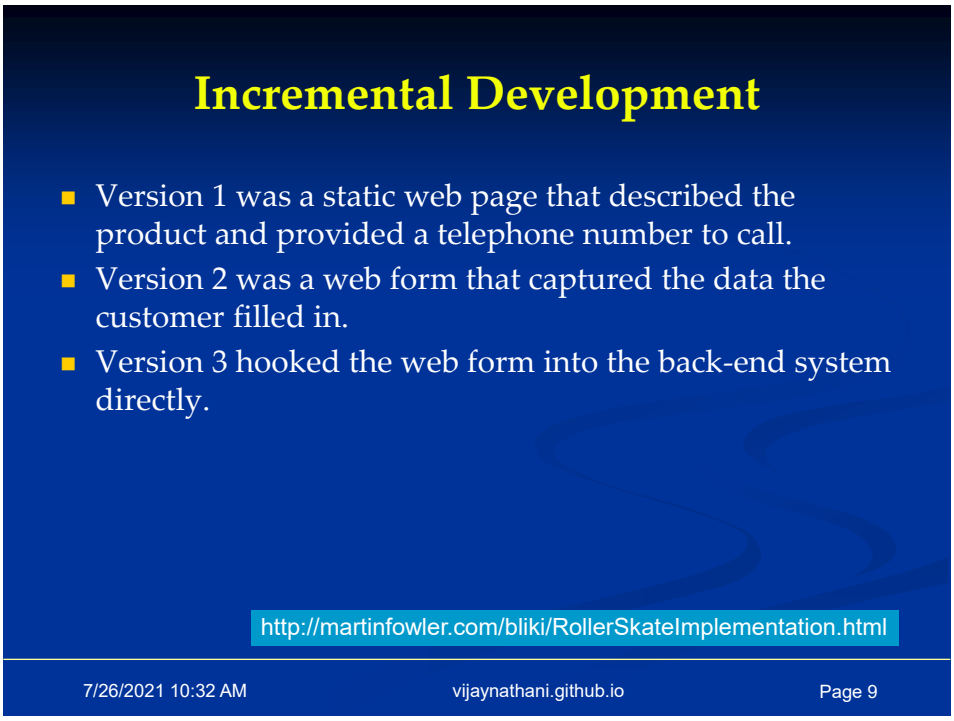


Planning development to add value quickly

A,B,C,D are components
A1,B1,C1,D1 add 25% value for each component
Together, these 4 add 60% of overall value to customer
Do them first



Only the first one.



A key property of agile development is figuring out how to make a system go live with a small subset of features. We build software for the business value it offers, the quicker we go live, the faster we get at least some of that business value.

My colleague Dave Leigh-Fellows told me one of my favorite examples of this kind of thinking. It came when we has working for a brokerage firm. They had a new kind of product that they wanted to get into the market. The full software support for this was a web page that the customer filled in that generated the necessary transactions against the back-end system. But Dave came up with a way to get the product into the market faster than that.

Version 1 was a static web page that described the product and provided a telephone number to call. Some temporary staff then spoke to the customer and entered the information into the back end system.

Version 2 was a web form that captured the data the customer filled in. However this version didn't load that data into to the back end system. Instead the web form generated a fax. They hired some more temps to get the orders from the fax machine to the people that keyed the information into the back end system. Since the fax machines were a bit of a distance away, this is where the roller-skates came in.

Version 3 hooked the web form into the back-end system directly.

The first two versions may not have been the most elegant solutions ever conceived, but they did get the product into the market much more quickly. I've not come across any other examples of iterative development that use roller-skates, but that may be more due to lack of imagination rather than lack of need.

In Building construction: You know what needs to be done, Little discovery is taking place.

In Software: We don’t know the details, Much of the time is spent in discovering What the customer wants and how to buid it. Building it takes comparatively lesser time.

Why isn't building a building a valid metaphor for software development? The short answer to this question is that buildings are largely static and software is not. Buildings are very expensive to modify after they have been built, and indeed the entire process used to design and construct the building is intended to maximize the chances that the building will meet the customer's requirements and will, within some margin of error, stay within the desired budget. It's easy to identify when a building is complete, but successful software never is. Software is cheap to modify, and any successful piece of software is virtually guaranteed to change over its lifetime. Actually, the software industry has trained its users into believing that major modifications are possible, if not inevitable and expected.

Coral Reef: a marine reef composed of the skeletons of living coral, together with minerals and organic matter

Coral: a marine organism that lives in colonies and has an external skeleton.

A coral reef is continuously evolving and changing.

Too many development teams develop software products as if they were building a building. They place too much emphasis on features of the building and trying to completely understand its requirements before it is built. The temptation is to finish the project or product and then to stand back and admire the creation. Inevitably, however, unanticipated changes come along in the form of user requests, bug fixes, and changes to other software or hardware that the program is dependent upon. The result is that over time, doors get added in odd places, walls get thinner and are moved around, cables get run through elevator shafts, and holes get added between floors or rooms. A mess accumulates that becomes increasingly hard to manage, and the greater the mess, the easier the temptation to add to it. Add in competitive pressures and a constantly changing ecosystem of underlying technology (operating systems, web browsers, databases, UI toolkits, etc.) and chaos ensues.

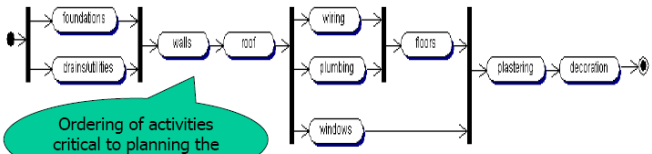
These projects are on the path to unsustainability. At some point the whole edifice will come tumbling down and become unmanageable. One of the main features will be an increasingly high cost of change where every change, no matter how minor, introduces a risk that something completely unrelated will break. The programmers will blame management, and management will blame everyone else. If the company can manage it, the application will be rewritten, only to start the clock ticking on the unsustainability cycle again. These projects aren't any fun, yet they are distressingly common in the software industry. The software industry is not slow-paced after all; laggards do not have high odds of survival.

Iterative Development

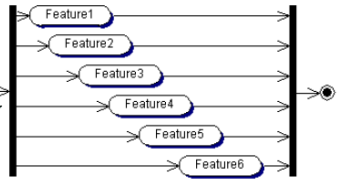
Very Small Exposure Requires Very Little Ceremony

Planning Methods contrasted

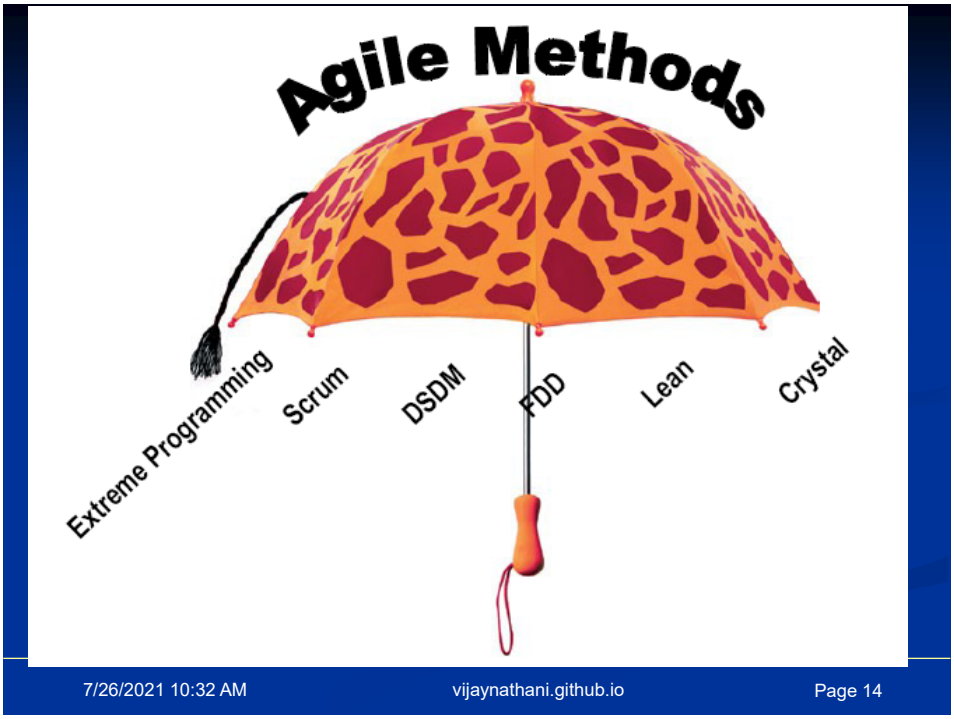
a) Critical Path Analysis



Ordering of activities critical to planning the building of a house



The dependencies in software are not as *concrete* as those in a physical building project



Ken Schwaber on MSF (Microsoft Solutions Framework), “[MSF for Agile] defines roles within the development team, so it isn’t cross-functional. It tells the team what to do, so it misses self-management. It defines what steps to take, so it isn’t empirical. It has increments that aren’t potentially shippable, so it isn’t lean. Lots of artifacts and no emerging list of requirements and architecture, so it is wasteful and not lean/agile.”

Removing many of the core principles that make Agile the most effective way to work isn’t the only issue. Microsoft chose to roll this process into their tools and the marketplace without independent peer review, without testing it with real teams, and with no comparative analysis on this new software development process vs. proven Agile methods such as Scrum. Should we expect better from the world’s largest software development tool vendor? Microsoft has enormous market presence, and as such has the opportunity to influence many billions of dollars of IT projects. Microsoft had the opportunity to make proven Agile methods the centerpiece of a tool suite that could help customers make the transition to much improved productivity. Instead Microsoft co-opted an important brand (Agile), but delivered a process that is missing key Agile principles and practices, and cannot provide the business value seen by teams adopting Scrum, XP, or other market tested and proven methods.

Not unchartered territory

- Agile practices are not new – since mid 90's
- Collection of proven practices proven
  - Scrum
  - Extreme Programming (XP)
- No longer a grassroots movement:
  - Wider adoption among Fortune 1000 companies
  - 75-80% of IT organizations have some Agile practices*
  - 15-25% have established a repeatable Agile process*

"Current State of Agile Methods Adoption", Gartner, December 2008