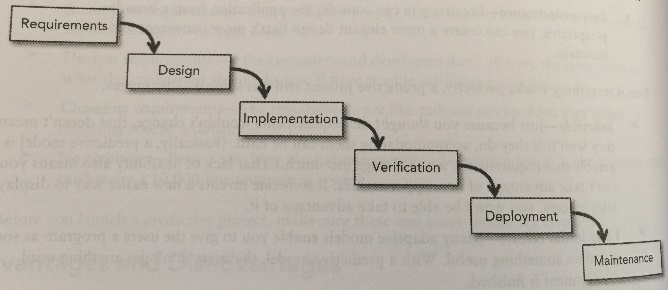
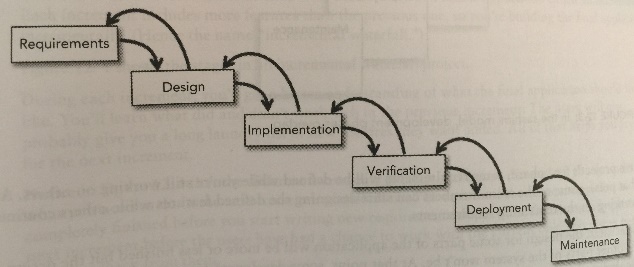
Software Engineering Models

**Predictive Models** – when the scope of the project is known and will not need to change.

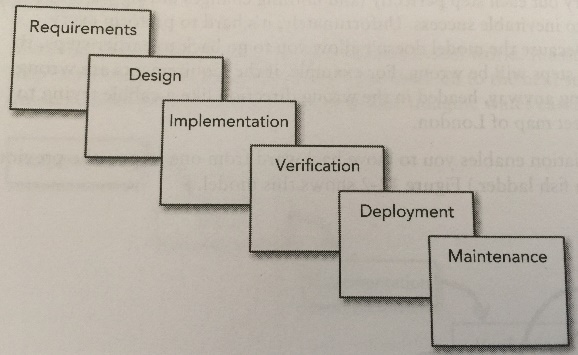
1. Waterfall – finish each step completely before moving on to the next (not very flexible)



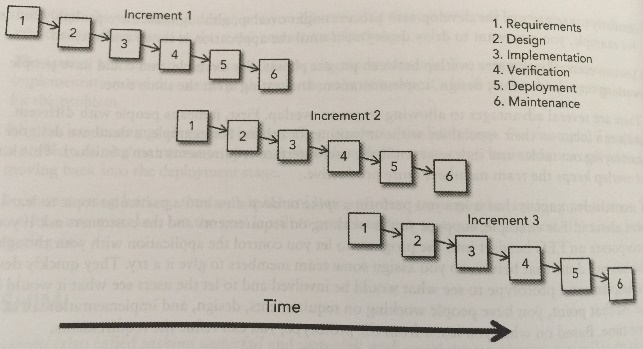
1. Waterfall with Feedback – same as waterfall, but can go up the waterfall when needed (but not advised)



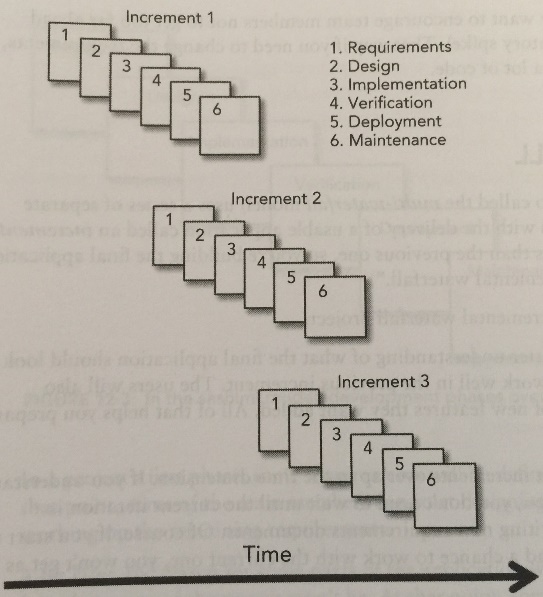
1. Sashimi – same steps of waterfall but you can move on to the next step before being completely done with the previous step.



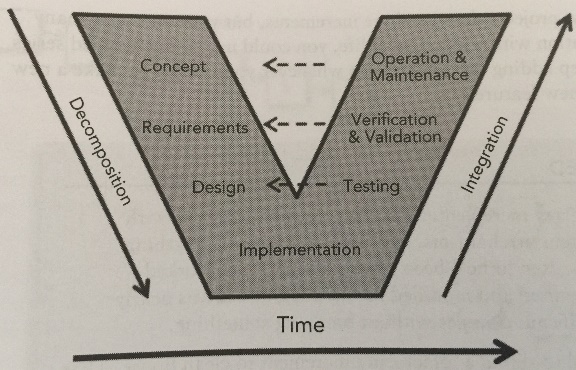
1. Incremental Waterfall – Start the next version before completing the current version. If you notice that your design/requirements aren’t right, you fix it in the next version not the current one.
   1. Incremental waterfall model



* 1. Incremental sashimi waterfalls

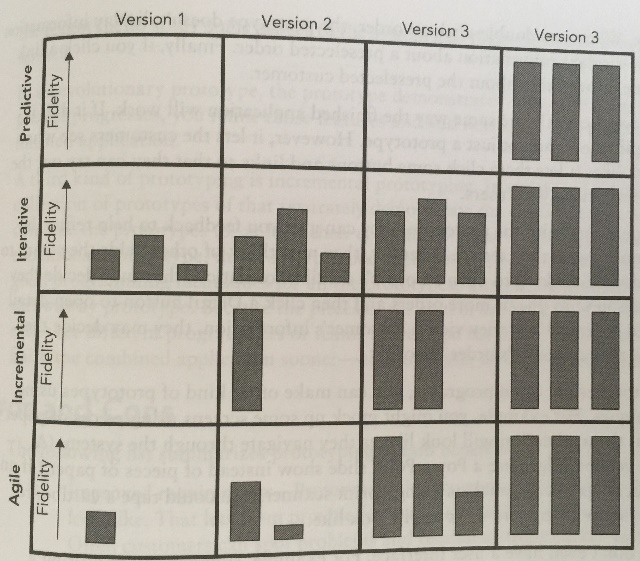


1. V-Model – basic idea of waterfall but look at it as a process of breaking the dream down into details and then building it back out to the dream.

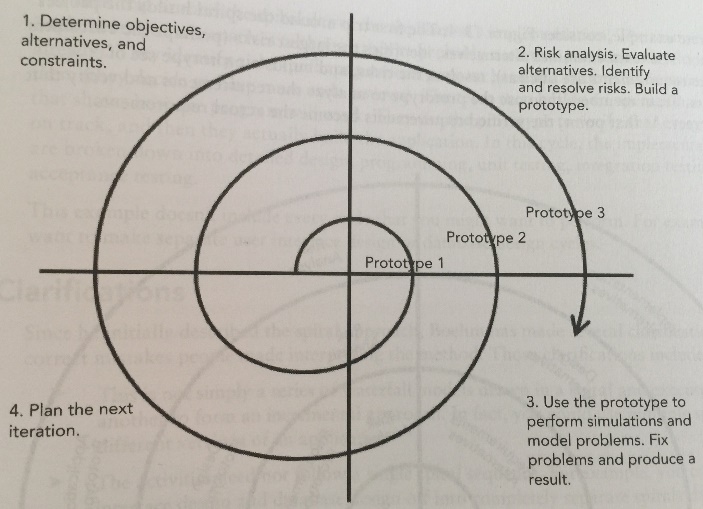


**Adaptive Models (iterative)** – when the scope of the project is likely to change during development.

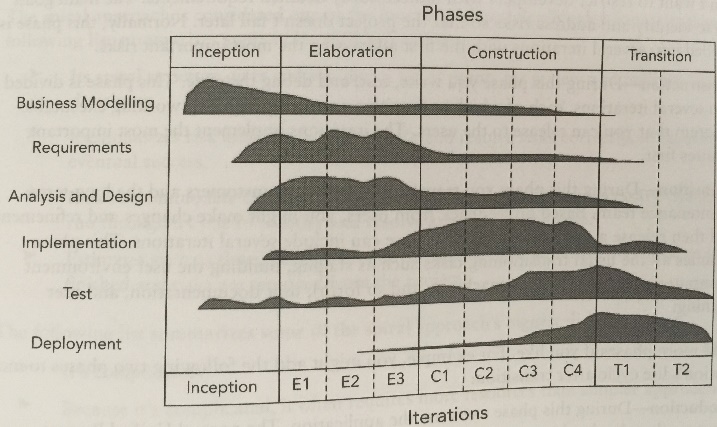
* Iterative (work on all features but at low fidelity) vs incremental (provide less features but with high fidelity) vs Agile (lowest features at low fidelity)



1. Prototypes
   1. Horizontal – give user feel for what it will look like (frontend) not how it will function.
   2. Vertical – no real UI but shows how program will work (backend)
2. Spiral – big focus is risk mitigation (not waterfall where you have to do 6 steps each time, can focus on a part of a single step or multiple steps each iteration). This gives the ability to make sure it is still on the right track after each iteration and the ability to scrap any given iteration.

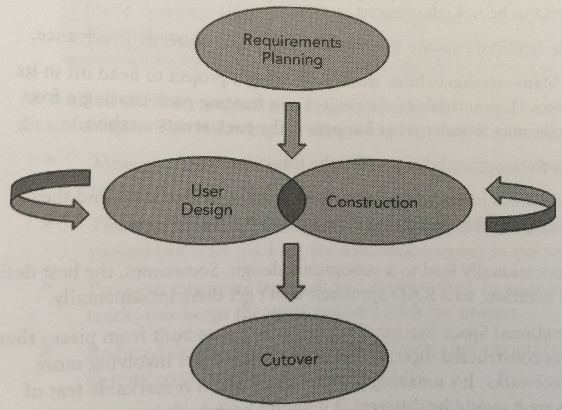


1. Unified Process – same idea as waterfall but different step naming and can work on multiple steps at the same time and out of order (i.e. testing the whole time). Also can have different number of iterations for each phase.

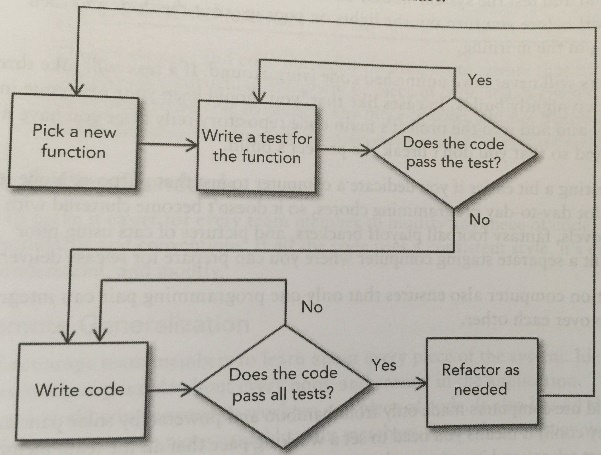


**Rapid Application Development (RAD)** – release features in increments

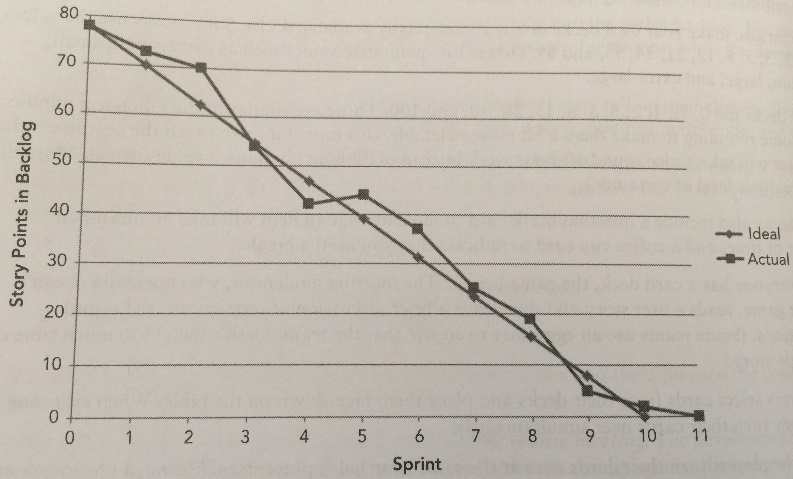
1. James Martin RAD – constant user feedback during development.



1. Agile
   1. Values of Agile development
      1. **Individuals and interactions** over processes and tools
      2. **Working software** over comprehensive documentation
      3. **Customer collaboration** over contract negotiation
      4. **Responding to change** over following a plan
   2. Agile Techniques
      1. Self-Organizing Teams – not assigned
      2. Communication – w/ users/client (almost constant)
      3. Incremental Development – build until enough to need a new release
      4. Focus on quality (since trying to do things so fast cant afford to have bugs)
2. Extreme Programming (XP)
   1. Roles
      1. Customer – defines requirements, makes sure meets user’s needs
      2. Tracker – monitors team progress
      3. Programmer – defines architecture and writes code
      4. Coach – helps team work effectively
      5. Tester – helps customer write tests to look for missing requirements
      6. Administrator – keeps computers up to date
      7. \* can combine roles but programmer shouldn’t be combined with customer, tester or tracker
   2. Values
      1. Communication – for requirements (simple, frequent and use metaphors)
      2. Simplicity – simple design first and more complex only when needed
      3. Feedback – unit and integration tests
      4. Courage - to make simple, refactor, throw away code, provide feedback
      5. Respect – good code quality
   3. Practices
      1. Customer on site
      2. Planning game - stories on cards to decide how many to fit into that iteration
      3. Standup meetings – 15 min or less
      4. Frequent small releases
      5. Intuitive metaphors – for communication
      6. Keep designs simple
      7. Defer optimization – only optimize if it is needed (saves time)
      8. Refactor when necessary
      9. Give everyone ownership of code
      10. Use coding standards
      11. Promote generalization – everyone knows the whole system
      12. Pair programming
      13. Test constantly
      14. Integrate continuously – merge code into master and make sure it works
      15. Work sustainably – no more than 40 hour weeks (avoid burnout)
      16. Test-driven and test-first development.

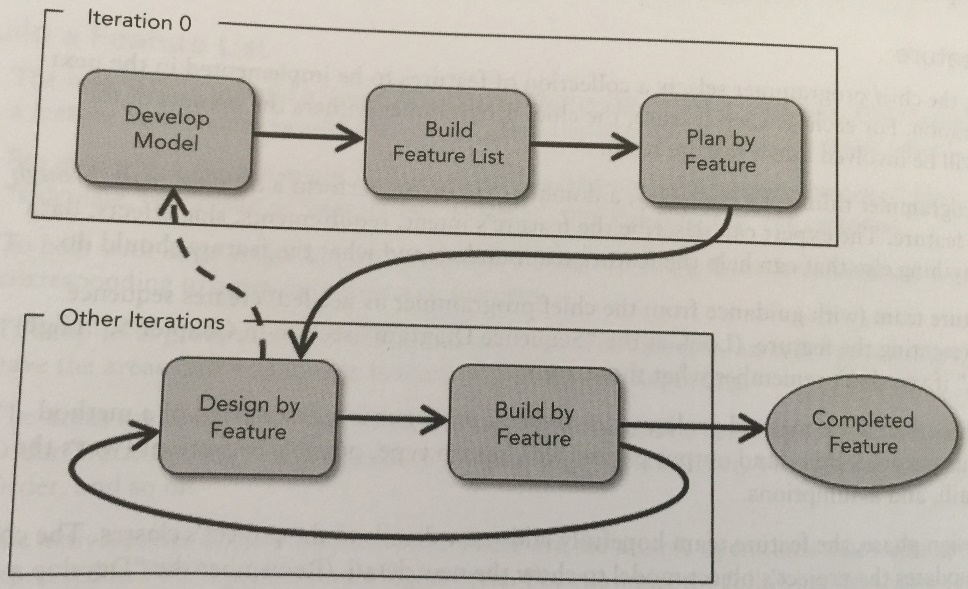


1. Scrum
   1. Roles
      1. Product owner – sets requirements/priority/plans/releases
      2. Team member
      3. Scrum master – removes obstacles for team (not a project manager because of self-organizing teams, but takes the leadership role)
   2. Scrum Sprints
      1. Whole project is broken down into series of sprints (smaller projects)
      2. Typically a sprint is between 1 week and 1 month
      3. Daily scrum (10-15 min each day)
         1. What did you do since last scrum?
         2. What do you hope to accomplish before next scrum?
         3. What obstacles do you see in your way?
      4. After Sprint Review meeting – lead by scrum master
         1. What went well and how can we make it happen again?
         2. What went poorly and how can we avoid that in the future?
         3. How can we improve the next sprint?
   3. Scrum Poker – game to determine how much work a particular task might be
      1. Use cards based on Fibonacci sequence (0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89) or could round (0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100)
         1. Also could include ?? card or some sort of card to indicate you need a break (coffee cup)
      2. Moderator leads – normally doesn’t play
         1. Reads user story and brief discussion of restrictions, risks, assumptions (can be timed to make sure doesn’t take too much time)
         2. All players decide how long they think the project will take and put that card face down
         3. All players turn over cards at same time
         4. Player with highest/lowest numbers are given soapbox to explain why they feel their estimate is correct.
         5. Repeat steps 2-4 until group reaches consensus and write down projected time for that task.
   4. Burndown – chart that shows amount of work remaining on a project as well as current progress based on scrum sprints

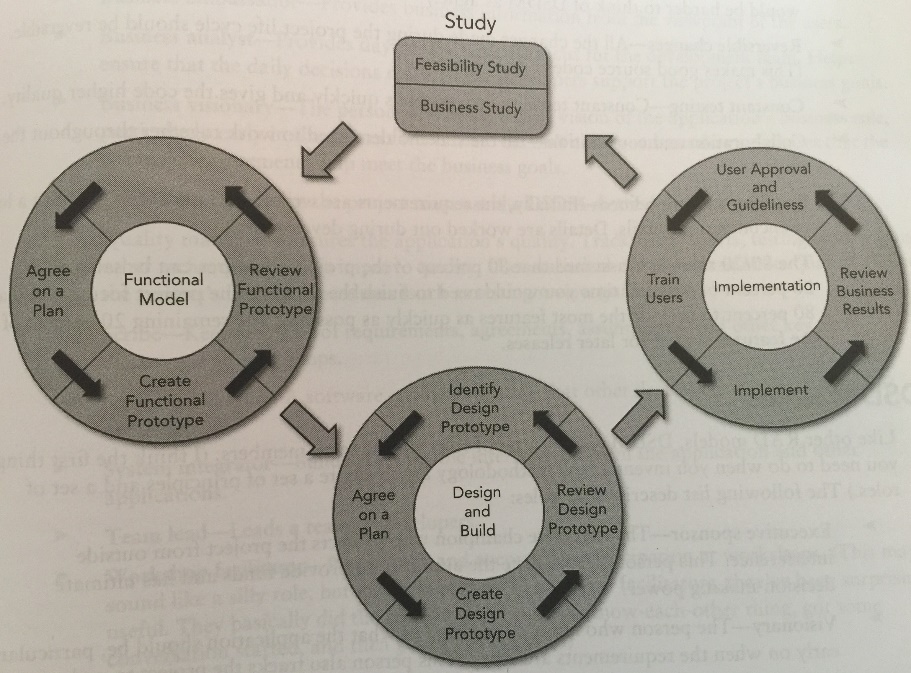


* 1. Velocity – amount of work team can perform during a sprint.

1. LEAN – don’t do anything that doesn’t directly contribute to the project (including unclear requirements, unnecessary features, unnecessary repetition, unnecessary meetings
2. Crystal Clear (1-6 member teams)
   1. Criticality
      1. Comfort
      2. Discretionary money
      3. Essential money
      4. Life
   2. Roles
      1. Sponsor – client/user
      2. Senior designer – makes design/technical decisions
      3. Programmer
3. Feature-Driven Development
   1. Dream up the whole system (features) at the beginning
   2. Then build them out one feature at a time until system is complete



1. Agile Unified Process (AUP) – Same as Unified Process but have several smaller releases instead of just one big release at the end.
2. Dynamic Systems Development Method (DSDM)
   1. Phases
      1. Pre-project – figuring out possible projects, approval, funding, etc.
      2. Project life-cycle



* + 1. Post-project – maintenance
  1. Principles
     1. Active user involvement
     2. Team empowerment – authority to make design decisions
     3. Frequent delivery
     4. Meet business needs – requirements
     5. Iterative and incremental – provide for quicker delivery/feedback
     6. Reversible changes
     7. Constant testing
     8. Collaboration and cooperation
     9. Requirements are refined
     10. The 80/20 rule – get 80% features in (takes 20% of time) and save last 20% (80% of time) for later releases to get features out quicker.
  2. Roles
     1. Executive sponsor - funder
     2. Visionary – clear vision of application
     3. Ambassador user – liaison between users and developers
     4. Technical coordinator – overall design/architecture
     5. Developer
     6. Tester
     7. Project manger

1. Kanban – Just-in-time (JIT) production methodology (not necessarily process)
   1. Practices
      1. Visualize workflow
      2. Limit work in progress (WIP) – help with task switching
      3. Enhance flow – when task is finished grab next-highest priority task
         1. Different from sprints because never doing multiple tasks at a time
   2. Kanban Board – want as few items in shaded zones as possible at a given time.

