**An Incremental Architecture Approach to Building Systems**

* 90% of applications are served by a monolithic approach.
* It is much better to start simple and evolve later on.
* Have a start phase, then a scaling phase and then an optimization phase.
* Starting phase: small group of people working towards a horizontal timeline of 3-4 months.
* Scaling phase: Maintain the application with a much larger group of people.
* Optimization phase: Keep the same set of functionalities but use less resources in a longer horizontal timeline of 2-5 years.

**Basics of the Unix philosophy**

* Write programs that does one thing well and work together. Also write programs to handle text streams
* Modularity: Write something simple that can be easily connected.
* Clarity: Write code that is clear rather than complicated and likely to break.
* Composition: Connect programs together well. One output becomes another program’s input.
* Separation: Separate policy from mechanism. Eg: Separate different processes clearly.
* Simplicity: Do not add complexity unless absolutely necessary.
* Parsimony: Only write a big program if a small program does not suffice. (Can hurt maintainability)
* Transparency: Design for visible code in order to make debugging easier. Transparency MATTERS.
* Robustness: Must design programs to handle unexpected inputs or changes.
* Representation: Make programs that handles various forms of data.
* Least Surprise: Do the least surprising thing when it comes to interface design.
* Silence: A program should not be activated unless it is told to do so.
* Repair: Write software to cope and handle incorrect inputs as gracefully as possible; otherwise diagnose the problem quickly and make changes.
* Economy: Use less programmer time and make more use of machine time.
* Generation: Write programs that allows for the automation of writing more programs.
* Optimization: Get every function working before hoping to optimize it.
* Diversity: There are multiple methods to solving one single problem.
* Extensibility: Design for the future, make your programs robust and scalable.

**Design in Construction**

* Design problems can initially be difficult to discern until further down the line.
* Designs can involve tradeoffs at deadline approaches.
* Designs can be nondeterministic as different people can come up with vastly different solutions.
* Design choices can evolve as a project is being churned out.
* Managing complexity is essential as uncontrolled complexity can lead to the failure of a project.
* One must minimize complexity in solving problems and to keep accidental complexity from proliferating.
* There should be 5 levels to design:

1. Software System: The entire system.
2. Division into subsystems: Databases, UI, business rules, backend etc.
3. Division into classes: Create classes that can be used to build routines.
4. Division into routines: Methods to run algorithms or to process data structures.
5. Internal Routine Design: Laying out the functionality of each individual routine.