- Part II - Starting with the Bricks: Programming Paradigms
  - 3. Paradigm Overview
    - Structured programming imposes discipline on direct transfer of control
      - Removed
        - goto
      - Added
        - if/then/else and do/while
    - Object-oriented programming imposes discipline on indirect transfer of control
      - Removed
        - function call stack frame
        - nested functions
        - function pointers
      - Added
        - constructors for a class
        - instance variables on heap
        - methods
    - Functional programming imposes discipline upon assignment
      - Removed
        - assignments
      - Added
        - lambda-calculus
        - immutability
    - ⚠️ Nothing else can be removed and no other paradigms can exist
      - ❓ What about Logic programming languages such as Prolog? Remove functions
      - ❓ What about scripting programming languages such as JavaScript, Python, Ruby? Remove compilation
      - ❓ What about visual programming, low code, no code? Remove the code
      - ❓ What about ML? Remove predefined logic
      - ❓ Quantum computers?
  - 4. Structured Programming
    - Dijkstra
      - first "programmer" and inventor of structured programmer
    - Böhm and Jacopini proved: all programs can be constructed from just three structures
      - sequence
      - selection
      - iteration
    - Dijkstra tried to use formal mathematical proofs for programs
      - But failed - it is too hard
    - Science proofs are more rational
      - Such proofs are falsifiable (testable) but not provable
      - 👍 If we can't prove statement is false then we consider it true
      - Testing shows the presence, not the absence, of bugs
      - We show correctness by failing to prove incorrectness, despite our best efforts
    - 👎 Programs that use unrestrained goto cannot be deemed correct no matter how many tests are applied to it
    - Functional decomposition - one of best practices
  - 5. Object-Oriented Programming
    - Dahl and Nygaard moved the function call stack frame to the heap and invented OO
    - Encapsulation?
      - 👎 But perfect encapsulation was in C and it was then weaken in C++ and weaken in Java/C# even more
    - Inheritance?
      - 👎 But C had kind of inheritance - via enclosing parent structure fields
      - 👍 However in OO upcasting is implicit - so - more convenienient
    - Polymorphism?
      - 👎 But it is implemented via vtable with consists of pointers to functions which were used long time before
      - 👍 However OO made it more convenient and eliminates danger
      - 👍 OO enables the plugin architecture!
    - Dependency inversion
      - Using interfaces
        - Lower level is dependent on higher level
      - Polymorphism means that any code dependency can be inverted
        - 👍 OO languages has absolute control over direction of code dependencies
    - Separate components without dependency
    - Independent deployability
    - Independent developability
  - 6. Functional Programming
    - Lambda-calculus was invented by Alonzo Church
    - Variables do not vary!
      - All race conditions, deadlock conditions, and concurrent update problems are due to mutable variables
    - 👍 Segregation of Mutability
      - good architecture practice
    - 👍 Event Sourcing
      - We store the transactions, but not the state
      - When state is needed we count it
        - Of course we can calculate and save intermediate states