The **SW Product Life Cycle (PLC)** is a framework by which an organization manages the development of its products from inception to EOL. The 4 key phases of PLC are:

- Exploration
- Planning
- Development
- Refresh

In the six phase model we have:

- Requirements Gathering
- Requirements Specification
- Architecture Design
- Detailed Design
- Implementation
- Validation and Verification

The key for SWE models is to have one that is communicated and used consistently. So that progress can be mapped and tracked.

Instruction Set Architecture (ISA) is the function specification for processor designers. It specifies the type of operations that can be performed and in what order they are performed. It specifies, where an operand will be stored (accumulator, stack, register). It specifies number of operands per instructions. The location where operands are stored and how they are placed. And the type and size of operands. In essence, it is a very low-level piece of software that handles exceptions, interrupts, and HW specific services. A collection of all the operations the HW is capable of performing. It is the boundary between running SW and hosting HW.

Five business and technical goals for large SW projects are as follows:

- Cost
    - Must be within the bounds of estimated space, money, and time.
- Portability
    - Must use widely available tools, and languages. SW involving a single language is best.
- Robustness
    - No bugs, managing and repair strategy.
- Legality
    - Have rights to all tools and establish clear rights to your created SW
- Backup
    - Define time between 2 complete backups if recovery is needed. Cost is known in case of total loss.

Typical SWE Challenges

- Legacy Systems
    - Old valuable system must be maintained and updated.
- Heterogeneity
    - Systems are distributed and include a mix of HW and SW.
- Delivery Time
    - Increasing pressure for SW delivery time.
- Other issues: Meeting requirements, staying within budget, documentation, on-time delivery, and inefficient use of resources.

Moore's Law is an observation made by Gordon Moore (co-founder of Intel) that the number of transistors per inch squared on an integrated circuit has doubled every year since its invention. Moore predicted that this trend would continue for the foreseeable future. This law has held true for decades. However, due to physical limitations in the process of manufacturing this law starting to break down in 2010 and the time-period has grown to 18 months.

The key message behind **Amdahl's law** is no matter how many CPUs you have, execution speed will be limited by software that is unable to be parallelized. Sequential software dominates execution speed.

Single accumulator architecture (stores results) was used in the earliest systems (1940s) and is sometimes called John Von Neuman's Computer or John Vincent Antasnoff's

Common Architecture Attributes:

- Main memory separate from the CPU
- Program instructions stored on main memory
- Data stored in memory, AKA Von Neuman's Architecture
- Instruction pointer (instruction counter, program counter)
- Von Neuman's memory bottleneck (everything on the same bus)
- Accumulator (1 or many) holds logical or arithmetic results.

General Purpose Register accumulates ALU results in X number of registers. It allows for register to register operations and is essentially a multi register extension of the Single Accumulator (SA) architecture.

Stack Machine Architecture (SMA) (Uniprocessor) is a zero address architecture since arithmetic operations require no explicit operand. Hence, no operand address. A pure stack machine has no registers and all operations are performed on memory. However, since memory is slow, stack machines are slow. The only advantage here is bits need not be wasted on memory locations for operations.

Pipelined Architecture (PA) (Uniprocessor) splits the ALU into separate sequentially connected units. This allows for multiple instructions at one moment in a uniprocessor. In essence, it starts an instructions works on it, starts another before finishing allowing for faster execution. If the pipeline is full all modules are currently executing an instruction. Branching (calls & returns) can slow down PA therefore branch prediction is used. Pipelined machine breaks instructions into sub operations and executes one sub operation of multiple sequential instructions in one step

RAW - Read after write dependence
WAR - Write after read dependence
WAW - Write after write dependence

Vector Architecture (VA) (Uniprocessor) uses registers that are split into index's that can load and store blocks of contiguous data. VA registers are essentially arrays. Note, may also have scalar registers.

Multiprocessor (MP) Architecture allows for multiple processors (cores) on a single chip. However, memory can only serve one chip at a time. Meaning other cores must wait until memory is free to get data.

Distributed Memory Architecture is a MP architecture where each processor has local memories. However, if data is needed from another processors local memory the core must wait until the other cpu returns the data. Yet, if data is on local memory this is very fast.

Systolic Array (SA) (MULTI) Architecture each processor has its own private memory with a network define by a systolic (pulse) pathway. Pathways can be bi-directional and can have a 2d or 3d topology. These pathways are high performance networks that send and receive data between CPU's.

Super Scalar Architecture (SSA) (Uniprocessor) replicates some operations in HW. May have multiple ALU's, FPU's but still is a uniprocessor architecture. Arithmetic operations can take place simultaneously.

VLIW machines have the potential to be extremely fast because they can perform multiple operations in a single instruction. However, a key issue with their operation is data dependencies. For example, the result of a Floating Point

ADD cannot be used as an operand for another operation in the same instructions. A VLIW machine is still limited by data dependencies. Sequentially executed code cannot be packed onto a single instruction.

SW Processes and **SWE models** are a structured set of activities that are used to develop a SW system. They involve:

1. Specification 2. Design 3. Validation 4. Evolution

A **SW Process Model** is an abstract representation of a SW process

The Waterfall Model separates distinct phases of specification and development. It is used to help organize and define SW development. Its main drawback being its difficulty accommodating change after the process is underway. The steps/phases are as follows:

- Requirements Analysis and Definition
- System and Software Design
- Implementation and Unit Testing
- Integration and System Testing
- Operation and Maintenance

Even though this model is supposed to be linear, IRL this is rarely the case. As we often hop back and forth between phases.

Incremental Development process is a SW development module. Development and delivery is broken into increments with each component delivering a piece of the required functionality. Each delivery of an increment generates early feedback which allows for correction and refinement of the product. This allows the SW to be more accurate to the customers needs.

The **incremental development process** is usually used in a deliberate situation. For example, a greedy company that wants quick delivery of SW. Therefore, SW is developed in increments that are presented for testing and feedback.

Data-Stream, Instruction Stream Classification
SISD - Single Instruction, Single Data Stream (PDP-11)
SIMD - Single Instruction, Multiple Data Stream (Array processors, solomon, Illiac IV)
MISD - Multiple Instruction, Single Data Stream (superscalar, pipelined, VLIW, EPIC)
MIMD - Multiple Instruction, Multiple Data Stream (true multiprocessor)

Reuse Oriented Development or **Component Based SWE** can be extremely wise. Chances are there is something similar that already exists. You are responsible for finding it. If so, can you acquire it legally? It it cost effective? Is it well documented? The idea is to cheapen and shorten the process of making software. However, be careful to verify that it does what you expect and that modifying it won't be more time consuming than developing from scratch.

Evolutionary Development takes into account **exploratory development** (work with customer to produce to evolve a final system from initial outline starting with well understood requirements) and **throw-away prototyping** (used to understand system requirements, starts with poorly understood requirements with a goal to clarify). It is good for small/medium systems or parts of larger ones and/or short lifetime software. Bad at long range process visibility and often requires special skills for fast prototyping.

Spiral Model can be used if there is a strong need to get early software working. Works well with a mature/organized team. It is a non-linear process where phases can be repeated in different stages of completeness. Risks are explicitly assessed and resolved in this process.

Integer Overflow occurs when the resultant from an operation is too large for the register. The result stored is a "cut-off" version of the actual value. It can be detected when the carry " in" bit is different from the carry "out" bit.

When integer overflow occurs most systems continue on. When **Floating Point overflow** occurs most systems crash.

A word in a modern x86 machine is 8 bytes long or 64 bits.

Word Addressable - Each unique word can be accessed independently of each other. If you need to access a single byte or bit you must use masking and shifting
Byte Addressable - Each unique byte is individually addressable
Bit Addressable - Each unique bit is individually addressable

Software Engineering (SWE) is the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software

History
Early computers 1940s, had hard-wired instructions they were easy to use, but inflexible! Single use! l Instead "stored program" architecture was needed, resulting in Von Neumann computer design. As a result, programming languages appeared in the 1950s (assembly, Fortran, Cobol). Trend continued with many additional languages into 1960s; luckily fewer languages nowadays! David Parnas argued 1960s for modularity, for information hiding 1970s. The Term "Software Engineering" evolved in the 1960s, popularly documented by Bauer.

Capability Maturity Model (CMM) used to evaluate the maturity level of SW projects.

1. Initial 2. Managed 3. Defined 4. Quantitatively Managed 5. Optimized

SWE crisis happened early during explosive evolution of the computer industry. Dijkstra coined this term in 1972. Causes of the crisis were linked to HW and SW complexity. Over Budget, Over-time, Inefficient, Low Quality, Failed to meet requirements, and Unmanageable or never delivered.

SWE was coined as a response to the chronic failures of large SW projects.

Code of SWE Ethics:

- Public
  - Software engineers shall act consistently with the public interest.
- Client and Employer
  - SWE's shall act in a manner that is in the best interests of client, employer. Consistent with the public interest.
- Product
  - SWE's shall ensure their products meet the highest professional standards possible
- Judgement
  - SWE's shall maintain integrity and independence in their professional judgement.
- Management
  - SWE managers shall subscribe to and promote an ethical approach to development and maintenance.
- Profession
  - SWE's shall advance the integrity and reputation of the profession consistent with the public interest.
- Colleagues
  - SWE's shall be fair and supportive of each other.
- Self
  - SWE's shall participate in lifelong learning regarding their practice and profession and shall promote an ethical approach to the practice of their profession.