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:** Analyze market, business, technology trends & opportunities which identify product solutions.
- **Planning:** Formalize next level of detail, including market requirements, decision times, product scope, usage, features.
- **Development:** Implement requirements defined in planning phase. Milestones are synchronization points to quantify progress.
- **Refresh:** Constitute further product revisions (or EOL) after initial product launch. May include updates to SW or HW and other.

**Five business & technical goals** for large SW projects & managers 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.
- **Legal and Ownership Issues**

**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.

**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.

**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

**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

**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. Cons, more manpower and resources expended to produce the numerous increments.

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

**Key Goals and Deliverables during a SWE project include:**

- Correct Function: That which works, must work correctly.
- Complete Function: All promised features must work.
- Acceptable, define speed must be defined.
- Low cost of development and maintenance.
- Portability to other computer environments.
- Complete documentation and high stability and reliability.

Testing and validation includes:

- **Component Testing**: Individual components tested independently. May be functions, objects, modules, and/or coherent groups.
- **Systems Testing:** Testing of functioning system as a while. Validating emergent properties is of particular importance.
- **Acceptance Testing:** Testing customer to verify that system meets needs. Should be as large as necessary and as small as possible.

**Requirements Engineering** is the SWE process of establishing the services the customer requires from SW, and constraints under which it operates.

**Requirements Engineering Process:**

1. Feasibility Study 2. Requirement Elicitations & Analysis
   3. Requirements Specifications 4. Requirements Validation.

**Programming & Debugging** involves 1. Locate Error 2. Design Error Repair 3. Repair Error 4. Re-Test Program.

**Validating** - Process of running SW to demonstrate a requirement.
**Verification** - Verify the solution works.
**Verification and validation (V&V)** is intended to show that a system conforms to specification and meets the requirements of the customer (end-user)

**CASE (Computer Aided SWE)** is a compute environment to help develop project, display milestones and models in a more disciplined way.

**Activity Automation includes:**

- Graphical Editor: For system model development
- Data Dictionary: To manage design entities
- Graphical UI Builder: For UI construction
- Debugger: To support program fault finding
- Automated Translator: To generate program version

**Capability Maturity Model (CMM)** used to evaluate the maturity level of SW projects. Imposed to reduce cost & time, increase quality, and produce a better world.

- **Initial:** Characterized as 'ad hoc' and sometimes chaotic. Few processes are defined and success depends on individual effort.
- **Repeatable:** Basic management processes are established. Process discipline is in place to repeat earlier successes.
- **Defined:** SW process for management and SWE is documented, standardized, & integrated into a standard SW process
- **Managed:** Detailed measures of SW process & products are quantitatively understood and controlled.
- **Optimized:** Continuous process improvement enabled by feedback and piloting innovative ideas & technologies.

Except for Lvl 1 each can be broken down into different **key process areas (KPA).** KPA identifies related activities that, when performed collectively, are used to achieve goals.

1. Commitment 2. Ability 3. Activity 4. Measurement 5. Verification

**General SWE activities are:**

• Specification • Design • Implementation • Validation and Debug • Evolution

Some key issues that affect computer architecture is as follows:

- **Memory Access is Slow** - processor operates at a much faster speed than the CPU resulting in slowdowns as the CPU waits for memory.
- **Events Tend to Cluster** - unrelated items concentrate into one small domain
- **Heat is Bad** - Temp must stay within a certain range, or bad.
- **Resource Replication** - Will not always solve the problem due to data dependencies.

**Component-Based SWE** is based on systematic reuse where systems are integrated from existing components or **COTS (Commercial off the shelf)** systems. Process Stages: 1. Component Analysis 2. Requirements Modification 3. System Design with Reuse 4. Development & Integration

A **requirements document** is the official statement required to be crafted by system developers. It should include user requirements and system requirements.
**Natural language** is sometimes used in requirements documents. The **benefit**, is accessibility. Allowing a broader spectrum of users to understand the document. **Cons** include: Lack of Clarity, Requirements confusion, requirements amalgamation, ambiguity.
**Non technical requirements** required for commercial SW:

1. Cost 2. Legality 3. Ethical 4. Privacy 5. Safety

//int c[SZ][SZ], a[SZ][SZ], b[SZ][SZ];

```
for( int row = 0; row < SZ; row++ ) {
    for( int col = 0; col < SZ; col++ ) {
       for( int k = 0; k < SZ; k++ ) {
          c[ row ][ col ] +=
             a[ row ][ k ] * b[ k ][ col ];
```

Some **real world non-technical** but **quantifiable parameters** of a major piece of SWE is the schedule, # of people, lines of code written, ease of use, and cost, number of customer interactions, quality of SW products, and added value for SW vendors & customers.

**Requirements documents** can be created using **form-based specification** (describes the function or entity by listing a description of the inputs/where they come from and where they go). **Mathematical specifications** (notations based on mathematical concepts). **Graphical** (tables, pictures, graphs). **Tabular** (concisely defines condition and action). The point of having these other methods is to more concisely, efficiently, and accurately describe the specifications of a project. Often used to supplement NL specification.

**Functional requirements** define what SW should do. **Non-Functional requirements** define requirements not related to SW itself but necessary to consider. (timing constraints, development process, standards). **Domain requirements** simply consider who should have access to it. **User requirements** describe functional and nonfunctional requirements in a way that they are understandable by a user who lacks detailed technical knowledge.

**Non-Functional Classifications:**
Product Requirements - (Efficiency, reliability, portability, usability)
Organisational Requirements - (Delivery, Implementation, Standards)
External Requirements - (Ethical, Legislative, Safety, Privacy, Interoperability)

**Production SW** can be measured with Cost, Schedule, manpower. The total cost is a result of these.

**Evolutionary Development** works with customers to evolve a final system from an initial outline spec. Starts with well understood requirements, adding new features as they are proposed.
**SW disasters happen.** This is fact. Here are a few examples.

- Therac-25: Radiation therapy machine failed due to a race condition. Had the potential to fire a high power electron beam into patient without proper shielding, 3 dead, in the 1980s.
- Y2K - Due to an imprecision in past programs, the year 2000 could've been interpreted wrong. Cost $500 billion to fix in 1999.
- Mars Climate Orbiter - SW that controlled orbiter thrusters used imperial units rather than metric as specified by NASA. Cost $125 million in 1998.

Be precise, be picky. When analyzing code your writing. One single bit can destroy your SW's intent.

Ascii to int

```
int ascii_to_int() {
    int result = 0;
    char c = getchar();
    while (c != '\n') {
       if (isdigit(c))
           result = result * 10 + (c - '0');
       c = getchar();
    } return result;
}
```

Even though a SW plan & schedule take extra time and money it provides numerous benefits. These are.

- Ensure quality
- Ensure product consistency
- Cost containment
- Maintain competitiveness
- Avoid disparate practices within SWE industry.

When running a C program you can define certain parameters during runtime. These parameters are stored in the **argv** string array. **argc** is an integer that counts how many are stored. **envp** is a string array that holds the sessions environment variables.

**Satisficing** is the acceptance of an available SW option as satisfactory, though not a perfect and complete solution. Cooked up by Herbet A. Simon in 1956.

The **quality of a SW product** are determined by: 1. Efficiency. 2. Maintainability. 3. Robustness. 4. Portability. 5. Ease of Use & Cost. 6. Complete & Correct Function.

**Moore's Law** is an observation by Gordon Moore that the number of transistors per in^2 on an integrated circuit has doubled every year since invention. This remained true for decades, but has moved to 18 months recently.

**Amdahl's Law** states that the execution speed will be limited by SW unable to be parallelized. Sequential SW dominates execution speed.

**It is not wise during the creation of a SW solution** to sneak in additional features. They could introduce new bugs, require further testing, and cost more. In essence, you spent time doing something extra when the product could've been done sooner.

**Instruction Set Architecture (ISA)** is the function specification for processor designers. It is a collection of all the operations HW is capable of performing. It is the boundary between running SW and hosting HW.

Matrix Inversion:

```
//N = SZ of the square matrix
void invert(int m[N][N]) {
    int temp[N][N];
    for (int row = 0; row < N; row++) {
        for (int col = 0; col < N; col++) {
            temp[row][col] = m[col][row];
        }
    }
    m = temp;
}
```

If you set out to create a product, do not specify its workings in a programming language as this makes the barrier for understanding it high, and completes the project you are specifying! Better to write it out in natural language so that the consumer can understand it.