Welcome to the Eight CBK Domain:

In this domain we cover:

- Security has for a long time been an afterthought in software development, we need to design it in and not bolt it on.
- Security in the software development lifecycle.
        - How we include security in the software requirements of our SDLC.
    - Development environment security controls.
        - How security should be designed and planned in, and should be a requirement of our development.
    - Software security effectiveness
        - How we assess the effectiveness of our software security controls
    - Acquired software security impact.
        - How we test and secure software we buy from 3rd parties.

This chapter chapter is how we secure software as we develop it.
CBK 8 makes up 11% of the exam questions.

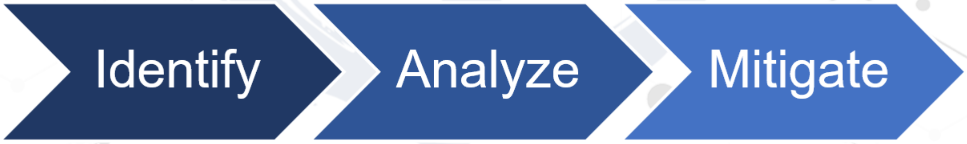
Designing Security into our Software:

- The more breaches and compromises, the more we see the move towards security being part of the scope of the software design project.
- We use software at our jobs, our personal lives, our homes, cars, power, water,...
- It is everywhere. And it has been, and still is, common to write functional code. Security is an afterthought, or not considered at all.
- A large part of our defense-in-depth is to protect our assets, but ultimately most of it is to protect our data/software.
- Software with security built in is much securer than software where it is added on later.
- It is common for programmers to make 15-50 mistakes per 1,000 lines of code. If using a programming maturity framework, we can lower that to 1 error per 1,000 lines of code.
- Most of the errors are not a vulnerability, or really a concern, but the more we use software in everything, the more critical the vulnerabilities become.
- Hacks have accelerated and stopped cars on highways, had planes change course (hacked through bad security on the in-flight entertainment), power grids, elections,...

- **Software-Defined Security (SDS):**
  - Security functions are removed from the hardware and become virtual network functions (VNFs).
  - SDS is software-managed, policy-driven, and governed security where most of our security controls (IDS/IPS, network segmentation, access controls, …) are automated and monitored by the software.

- **Auditing and Logging of Changes:**
  - We need to be clear on change and configuration management.
  - How changes are requested (Request, formal assessment, planning, design and testing, implementation, final assessment), the controls we have in place (request control, change control, and release control) and how they contribute to security.
  - How does our configuration management control the versions of software used in an organization.

- **Risk Analysis and Mitigation:**
  - With risk analysis and mitigation we identifying, assessing and mitigating the risks, understanding how they might affect our (software) project, and figure out what we can do to minimize the effects on our scope, schedule, cost, and quality.
  - Risks can be either opportunities or threats; they are scored on the likelihood of occurrence and impact on a project.



Programming Concepts:

- **Machine Code:**
  - Software executed directly by the CPU, 0's and 1's understood by the CPU.
- **Source Code:**
  - Computer programming language, written in text and is human understandable, translated into machine code.
- **Assembler Languages:**
  - Short mnemonics like ADD/SUB/JMP, which are matched with the full length binary machine code; assemblers convert assembly language into machine language. A disassembler does the reverse.
- **Compiler Languages:**
  - Translates the higher level language into machine code and saves, often as executables, compiled once and run multiple times.
- **Interpreted Languages:**
  - Similar to compiler languages, but interprets the code each time it is run into machine code.
- **Bytecode:**
  - An interpreted code, in intermediary form, converted from source code to interpreted, but still needs to be converted into machine code before it can run on the CPU.
- **Procedural Languages (Procedure-oriented):**
  - Uses subroutines, procedures and functions.
- **Object-oriented Programming (OOP):**
  - Based on the concept of objects, which may contain data, in the form of fields, often known as attributes, and code, in the form of procedures, often known as methods.
  - An object's procedures can access and often modify the data fields of the objects with which they are associated.
  - In OOP, computer programs are designed by making them out of objects that interact with one another.
- **4th Generation languages (4GL):**
  - Fourth-generation languages are designed to reduce programming effort and the time it takes to develop software, resulting in a reduction in the cost of software development.
  - Increases the efficiency by automating the creation of machine code.
  - Often uses a GUI, drag and drop, and then generating the code, often used for websites, databases and reports.
- **Programming Languages and Generations:**
  - **1st generation:** Machine Code
  - **2nd Generation:** Assembler languages
  - **3rd Generation:** C, C++, Java, Python, PHP, Perl, C#, BASIC, Pascal, Fortran, ALGOL, COBOL, …
  - **4th Generation:** ABAP, Unix Shell, SQL, PL/SQL, Oracle Reports, R, …
  - **5th Generation:** Prolog, OPS5, Mercury, …

- **CASE (Computer-Aided Software Engineering):**
  - Similar to and were partly inspired by computer-aided design (CAD) tools used for designing hardware products.
  - Used for developing high-quality, defect-free, and maintainable software.
  - Often associated with methods for the development of information systems together with automated tools that can be used in the software development process.
  - **CASE software** is classified into 3 categories:
    - **Tools** support specific tasks in the software life-cycle.
    - **Workbenches** combine two or more tools focused on a specific part of the software life-cycle.
    - **Environments** combine two or more tools or workbenches and support the complete software life-cycle.
- **Top-Down Programming:**
  - Starts with the big picture, then breaks it down into smaller segments.
  - An overview of the system is formulated, specifying, but not detailing, any first-level subsystems.
  - Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements.
  - Procedural programming leans toward Top-Down, you start with one function and add to it.
- **Bottom-Up Programming:**
  - Piecing together of systems to build more complex systems, making the original systems a sub-system of the overarching system.
  - The individual base elements of the system are first specified in great detail, they are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed.
  - OOP leans tends toward Bottom-Up, you start by developing your objects and build up.

- **Software Release:**
  - **Open source:**
    - We release the code publicly, where it can be tested, improved and corrected, but it also allows attackers to find the flaws in the code.
  - **Closed Source:**
    - We release the software, but keep the source code a secret, may be sound business practice, but can also be security through obscurity.
  - **Proprietary Software:**
    - Software protected by intellectual property and/or patents, often used interchangeably with Closed Source software, but it really is not. It can be both Open and Closed Source software.
    - Any software not released into the public domain is protected by copyright.

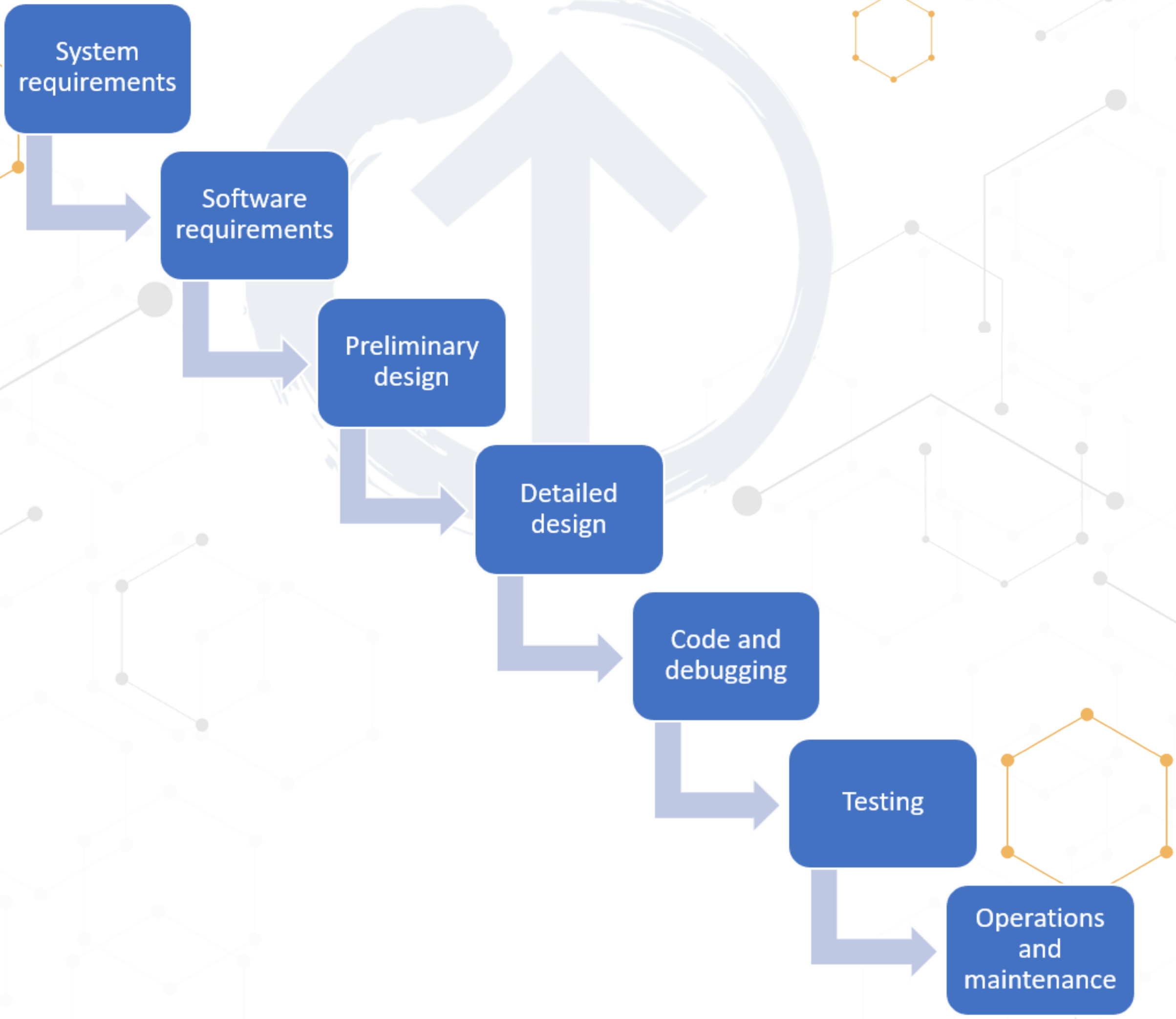
    - **Free Software:**
        - **Freeware:**
            - Actually free software, it is free of charge to use.
        - **Shareware:**
            - Fully functional proprietary software that is initially free to use.
            - Often for trials to test the software, after 30 days you have to pay to continue to use.
        - **Crippleware:**
            - Partially functioning proprietary software, often with key features disabled.
            - The user is required to make a payment to unlock the full functionality.
    - **EULAs (End-User License Agreements):**
        - Electronic form where the user clicks "I agree" to the software terms and conditions while installing the software.

- **Software Licenses:**
    - Open source software can be protected by a variety of licensing agreement.
        - **GNU (General Public License) also called GPL:**
            - Guarantees end users the freedom to run, study, share and modify the software.
            - A copyleft license, which means that derivative work can only be distributed under the same license terms.
        - **BSD (Berkeley Software Distribution):**
            - A family of permissive free software licenses, imposing minimal restrictions on the use and redistribution of covered software.
            - This is different than copyleft licenses, which have reciprocity share-alike requirements.
        - **Apache:**
            - Software must be free, distribute, modify and distribute the modified software.
            - Requires preservation of the copyright notice and disclaimer.

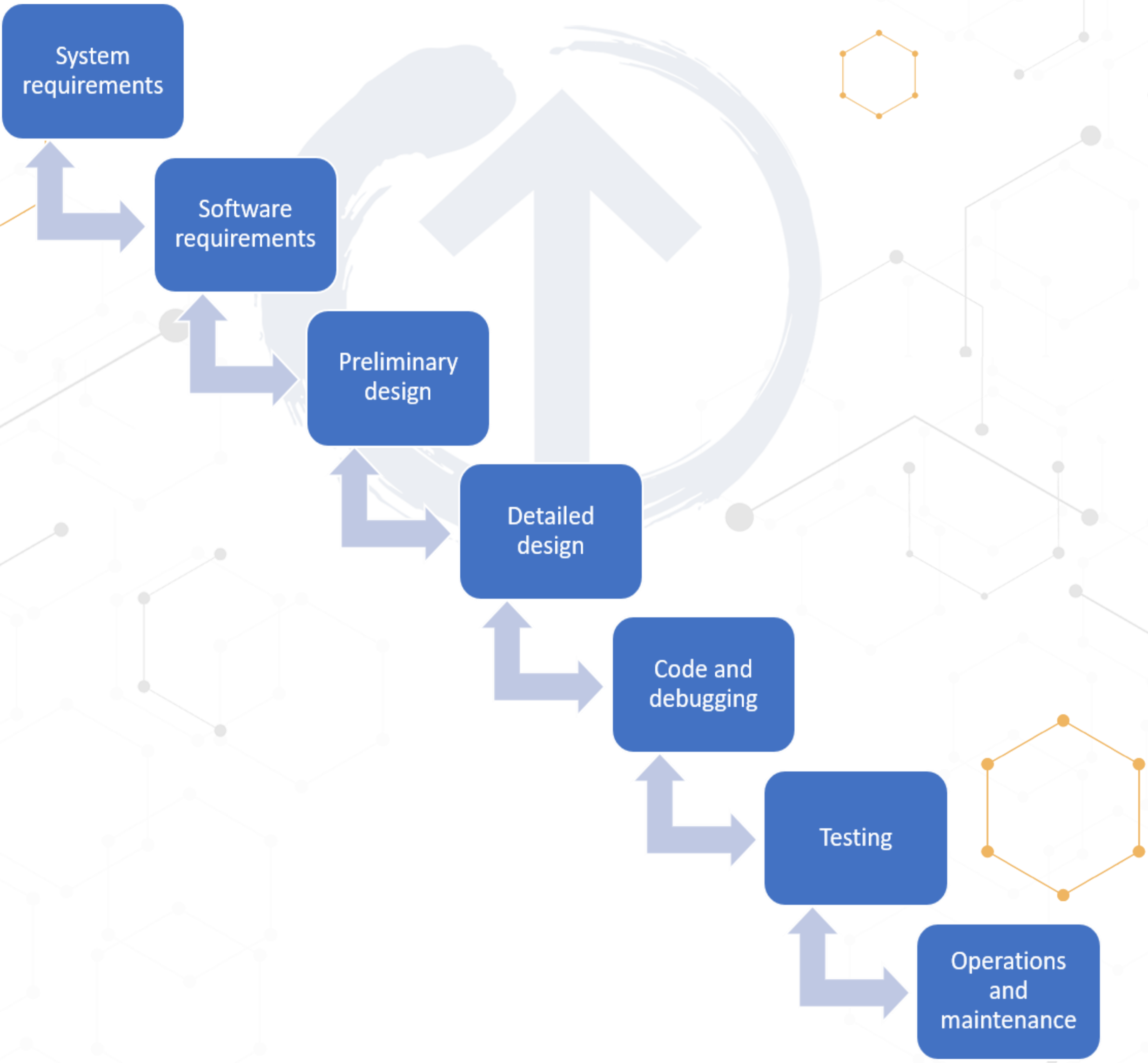
Software Development Methodologies:

- There is a wide range of software development methodologies used today.
- In the past the Waterfall method was widely used, it is a very linear process, and does not work very well with the iterative nature of software development.
- To remedy that problem other methods were developed Spiral, Sashimi, Agile and Scrum.
- The individual phases are different from organization to organization, understand how each methodology works and the phases flows.

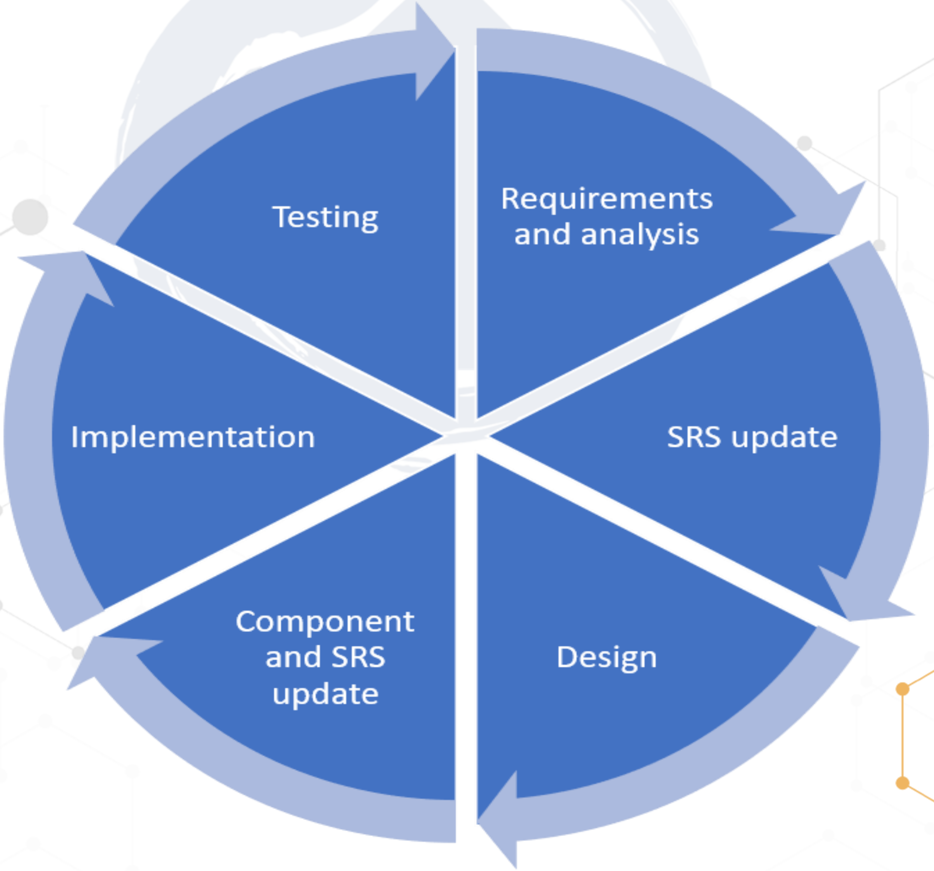
- **Waterfall:**
  - Very linear, each phase leads directly into the next.
  - The unmodified waterfall model does not allow us to go back to the previous phase.



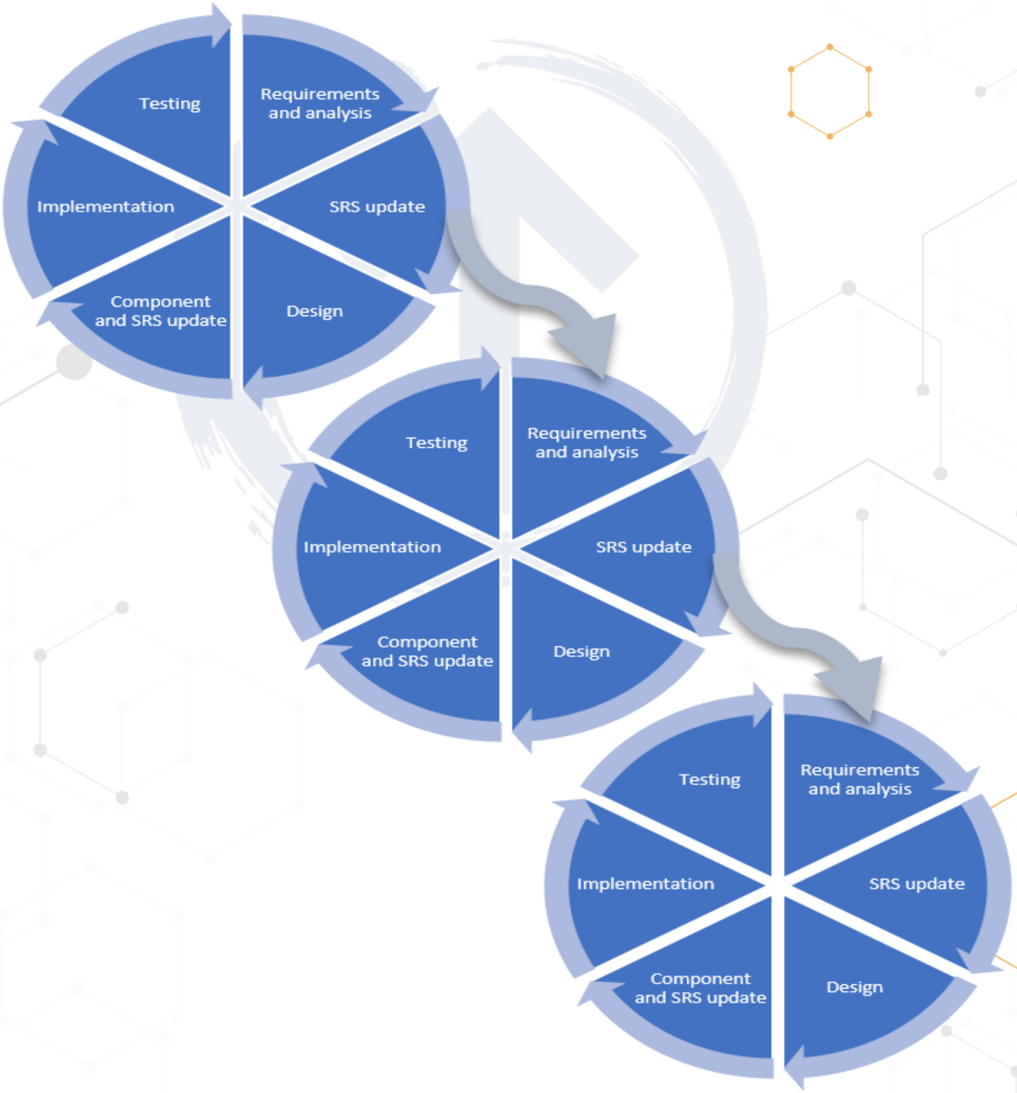
- **Sashimi Model** (Waterfall with overlapping phases):
  - Similar to waterfall, but we always have 2 overlapping phases, if we close one phase, we add the next phase.
  - The modified waterfall model allows us to go back to the previous phase but no further.
- **Agile Software Development:**
  - Describes a set of values and principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.
  - Uses adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.
  - There are many types of agile, for the exam know the flow.



- **Manifesto for Agile Software Development:**
  - **What is valued in the manifesto:**
    - Individuals and Interactions more than processes and tools.
    - Working Software more than comprehensive documentation.
    - Customer Collaboration more than contract negotiation.
    - Responding to Change more than following a plan.
  - **The twelve principles in the manifesto:**
    1. Customer satisfaction by early and continuous delivery of valuable software.
    2. Welcome changing requirements, even in late development.
    3. Working software is delivered frequently (weeks rather than months).
    4. Close, daily cooperation between business people and developers.
    5. Projects are built around motivated individuals, who should be trusted.

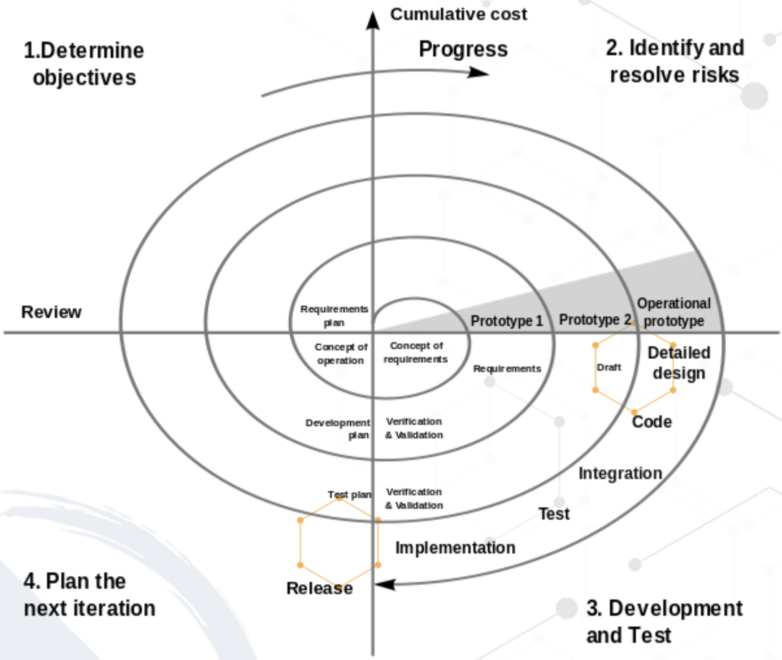


6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the primary measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.



- **Scrum:**
  - Scrum is a framework for managing software development. Scrum is designed for teams of approximately 10 individuals, and generally relies on two-week development cycles, called "sprints", as well as short daily stand-up meetings.
  - The three core roles in the Scrum framework.
    - **The Product Owner:**
      - → Representing the product's stakeholders, the voice of the customer, and is accountable for ensuring that the team delivers value to the business.
    - **Development Team:**
      - → Responsible for delivering the product at the end of each sprint (sprint goal).
      - → The team is made up of 3–9 individuals who do the actual work (analysis, design, develop, test, technical communication, document, etc.).
      - → Development teams are cross-functional, with all of the skills as a team necessary to create a product increment.
    - **Scrum Master:**
      - → Facilitates and accountable for removing impediments to the ability of the team to deliver the product goals and deliverables.
      - → Not a traditional team lead or project manager but acts as a buffer between the team and any distracting influences.
      - → The scrum master ensures that the Scrum framework is followed.
- **XP (Extreme Programming):**
  - Intended to improve software quality and responsiveness to changing customer requirements.
  - Uses advocates frequent releases in short development cycles, intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.
  - **XP uses:**
    - Programming in pairs or doing extensive code review.
    - Unit testing of all code.
    - Avoiding programming of features until they are actually needed.
    - Flat management structure.
    - Code simplicity and clarity.
    - Expecting changes in the customer's requirements as time passes and the problem is better understood.
    - Frequent communication with the customer and among programmers.

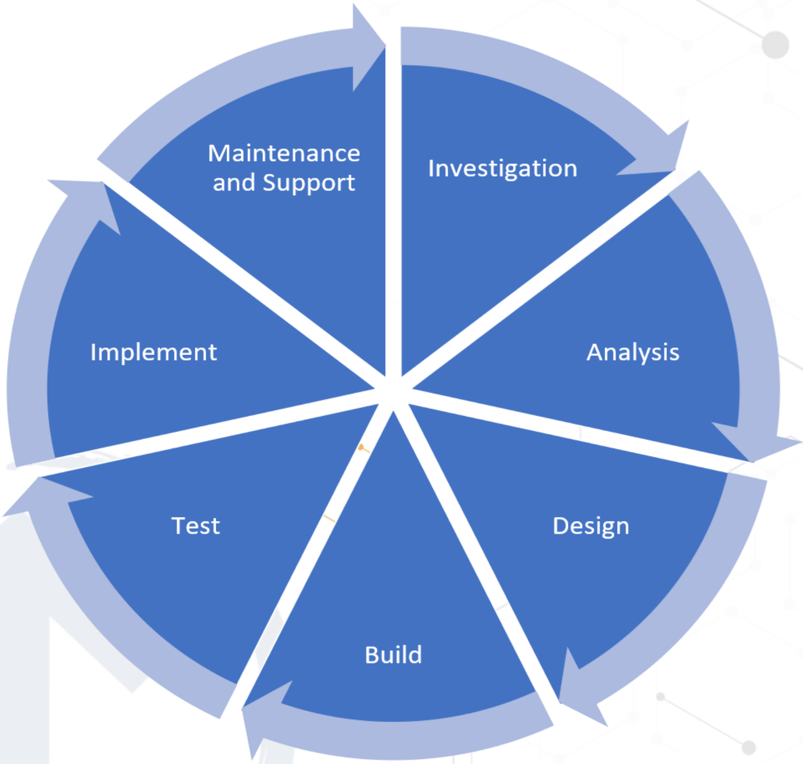
- **The Spiral Model:**
  - A risk-driven process model generator for software projects.
  - The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.
  - A software project repeatedly passes through these phases in iterations (called Spirals in this model).
  - The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed.
  - Each subsequent spiral builds on the baseline spiral.



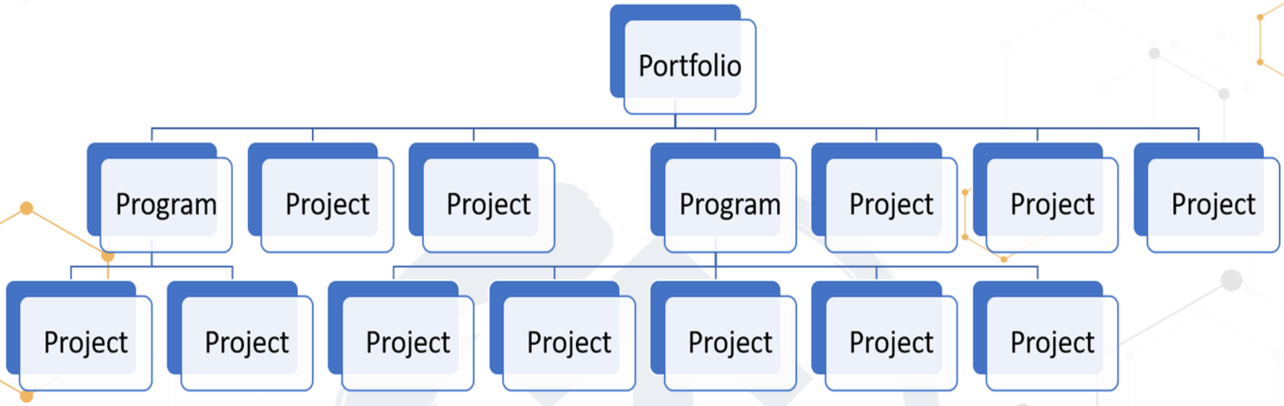
- **RAD (Rapid Application Development):**
  - Puts an emphasis on adaptability and the necessity of adjusting requirements in response to knowledge gained as the project progresses.
  - Prototypes are often used in addition to or sometimes even in place of design specifications.
  - Very suited for developing software that is driven by user interface requirements.
  - GUI builders are often called rapid application development tools.

- **Prototyping:**
  - Breaks projects into smaller tasks, creating multiple prototypes of system design features.
  - A working model of software with some limited functionality, rather than designing the full software up front.
  - Has a high level of customer involvement, the customer inspects the prototypes to ensure that the project is on track and meeting its objective.

- **SDLC (Software Development Life Cycle):**
  - The SDLC is not really a methodology, but a description of the phases in the life cycle of software development.
  - These phases are (in general), investigation, analysis, design, build, test, implement, maintenance and support (and disposal).
  - Can have security built into each step of the process, for the exam it always does.
  - If an answer about SDLC does not list secure or security, it would be wrong and can be eliminated.
  - Has a number of clearly defined and distinct work phases which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems.
  - The aim is to produce high-quality systems that meet or exceed customer expectations, based on customer requirements, by delivering systems which move through each clearly defined phase, within scheduled time frames and cost estimates.
  - SDLC is used during the development of a project, it describes the different stages involved in the project from the drawing board, through the completion of the project.
  - All software development methodologies follow the SDLC phases but the method of doing that varies vastly between methodologies.
  - Many different SDLC methodologies have been created, Waterfall, Spiral, Agile, Rapid Prototyping,...
  - In Scrum project a single user story goes through all the phases of the SDLC within a single two-week sprint, where Waterfall projects can take many months or several years to get through the phases.
  - While very different they both contain the SDLC phases in which a requirement is defined, then pass through the life cycle phases ending in the final phase of maintenance and support.



- **Projects, Programs and Portfolios:**
  - A **project** is a temporary endeavor, with a finite start and end, that is focused on creating a unique product, service, or result.
  - A **program** is a collection of related projects. Like a project, a program is temporary, when the collection of projects are complete, the program is complete.
  - A **portfolio** is a collection of projects and programs that are managed as a group to achieve strategic objectives.

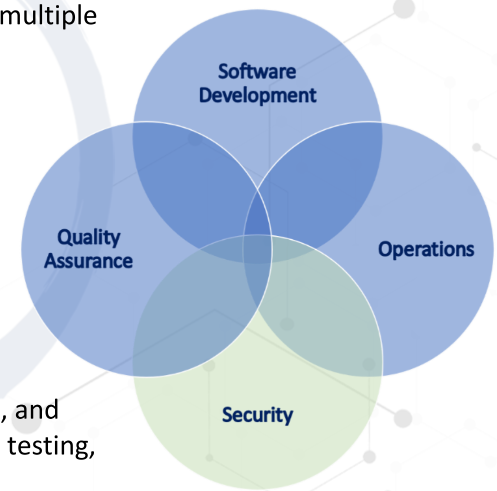


- **IPT (Integrated Product Team):**
  - A multidisciplinary group of people who are collectively responsible for delivering a defined product or process.
  - IPTs are used in complex development programs/projects for review and decision making.
  - The emphasis of the IPT is on involvement of all stakeholders (users, customers, management, developers, contractors) in a collaborative forum.
  - IPTs can be addressed at the program level, there may also be Oversight IPTs (OIPTs), or Working-level IPTs (WIPTs).
  - IPTs are created most often as part of structured systems engineering methodologies, focusing attention on understanding the needs and desires of each stakeholder.

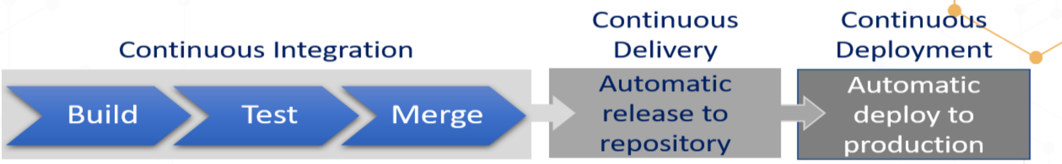
- **Source Code Escrow:**
  - The deposit of the source code of software with a third party escrow agent.
  - Escrow is typically requested by a party licensing software (the licensee), to ensure maintenance of the software instead of abandonment or orphaning.
  - The software source code is released to the licensee if the licensor files for bankruptcy or otherwise fails to maintain and update the software as promised in the software license agreement.
- **Source Code Repositories:**
  - Using public third party code repositories comes with some security concerns.
  - Other than the provider security, one of the most important controls is using multi-factor authentication.
  - File archive and web hosting facility where a large amount of source code, for software or for web pages, is kept, either publicly or privately.
  - They are often used by open-source software projects and other multi-developer projects to handle various versions. They help developers submit patches of code in an organized fashion.
- **API Security (Application Programming Interface):**
  - Allows an application to communicate with another application, operating systems, databases, networks,...
  - Many applications use API's, this could be to add super sign-on, integrate 2 applications, or many other things,...
  - They are a good example of how we integrate for better usability, but often security is overlooked.
  - API's are the cause of a number of recent high-profile website security breaches including Snap Chat, Pinterest and Instagram.
  - We covered the OWASP top 10 web vulnerabilities in domain 3.
  - OWASP also has an Enterprise Security API Toolkits project, which includes these critical API controls:
  - Authentication, Access control, Input validation, Output encoding/escaping, Cryptography, Error handling and logging, Communication security, HTTP security and Security configuration.
- **Software Change and Configuration Management:**
  - Earlier in this domain we covered how software development has a lifecycle, and in Domain 7 we covered configuration and change management.
  - Both change and configuration management are very applicable to our software development process, all the way from investigation/initiation to disposal of the software.
  - As with many of the concepts we cover they are to some extend logical, configuration management tracks changes to a specific piece of software where change management is all changes in the entire software development process.
  - NIST 80-128: Guide for Security-Focused Configuration Management of Information Systems uses these terms:
    - A Configuration Management Plan (CM Plan) is a comprehensive description of the roles, responsibilities, policies, and procedures that apply when managing the configuration of products and systems.

- The basic parts of a CM Plan include:
  - Configuration Control Board (CCB) – Establishment of and charter for a group of qualified people with responsibility for the process of controlling and approving changes throughout the development and operational lifecycle of products and systems, may also be referred to as a change control board.
  - Configuration Item Identification – for selecting and naming configuration items that need to be placed under CM.
  - Configuration Change Control – Process for managing updates to the baseline configurations for the configuration items.
  - Configuration Monitoring – Process for assessing or testing the level of compliance with the established baseline configuration and mechanisms for reporting on the configuration status of items placed under CM

- **DevOps:**
  - Cooperation between development, operations, and Quality Assurance.
  - Aligned with Agile, code is deployed rapidly, multiple times a day.
  - **CI/CD (Continuous Integration/Continuous Delivery):** The D in CD can also be referred to as Deployment or Development.
    - Code rollouts may happen many times per day, with that we need to rely on much more automation (integrating code repositories, deployment, infrastructure changes, software configuration management, and moving code between development, testing, production environments, …).

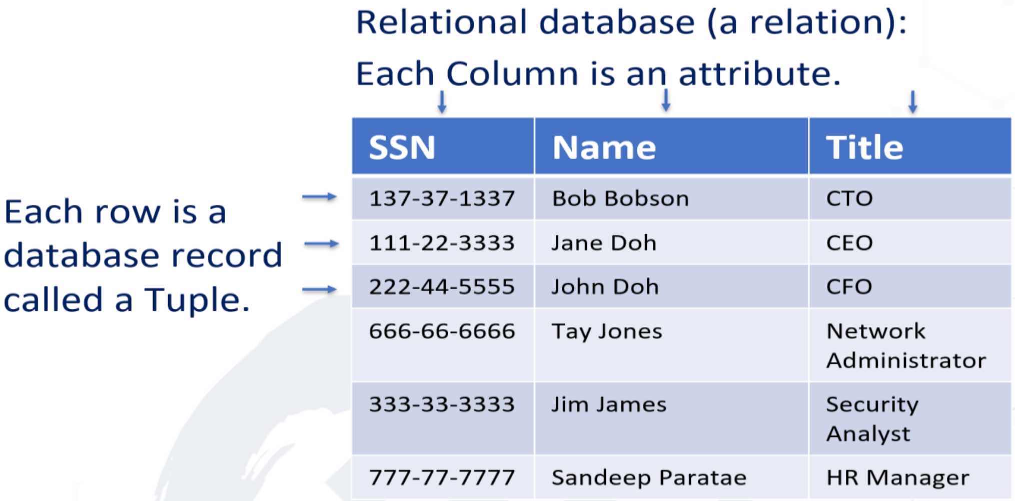


- **DevSecOps:**
  - Evolved from DevOps to add security into the process.
  - We want security to be integrated throughout the development process.
  - Security is not just added on later as an afterthought, it is designed in.



Databases:

- An organized collection of data.
- It is the collection of schemas, tables, queries, reports, views, and other objects.
- The data are typically organized to model aspects of reality in a way that supports processes requiring information. This could be modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.



Relational database (a relation):

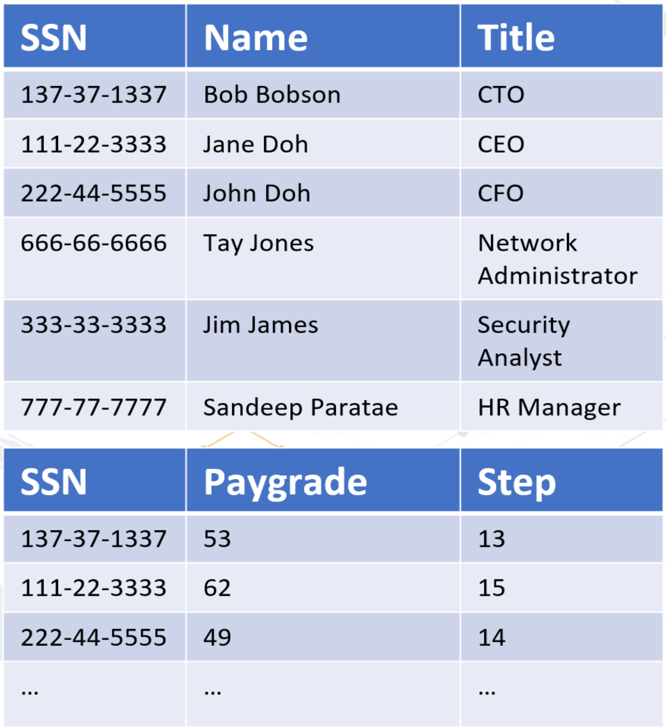
Each Column is an attribute.

Each row is a database record called a Tuple.

SSN	Name	Title
137-37-1337	Bob Bobson	CTO
111-22-3333	Jane Doh	CEO
222-44-5555	John Doh	CFO
666-66-6666	Tay Jones	Network Administrator
333-33-3333	Jim James	Security Analyst
777-77-7777	Sandeep Paratae	HR Manager

- **DBMS (Database Management System):**
  - A computer software application that interacts with the user, other applications, and the database itself to capture and analyze data.
  - A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.
  - MySQL, PostgreSQL, MongoDB, MariaDB, Microsoft SQL Server, Oracle, Sybase, SAP HANA, SQLite and IBM DB2.
- Database management systems are often classified according to the database model they support, the most common database systems since the 1980s have all supported the relational model as represented by the SQL language.
- A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model (the SQL version), which uses a table-based format.
- Common logical data models for databases include:
  - Navigational databases: Hierarchical database model, Network model, Graph database.
  - Relational model.
  - Entity–relationship model, Enhanced entity–relationship model.
  - Object model.
  - Document model.
  - Entity–attribute–value model.
  - Star schema.

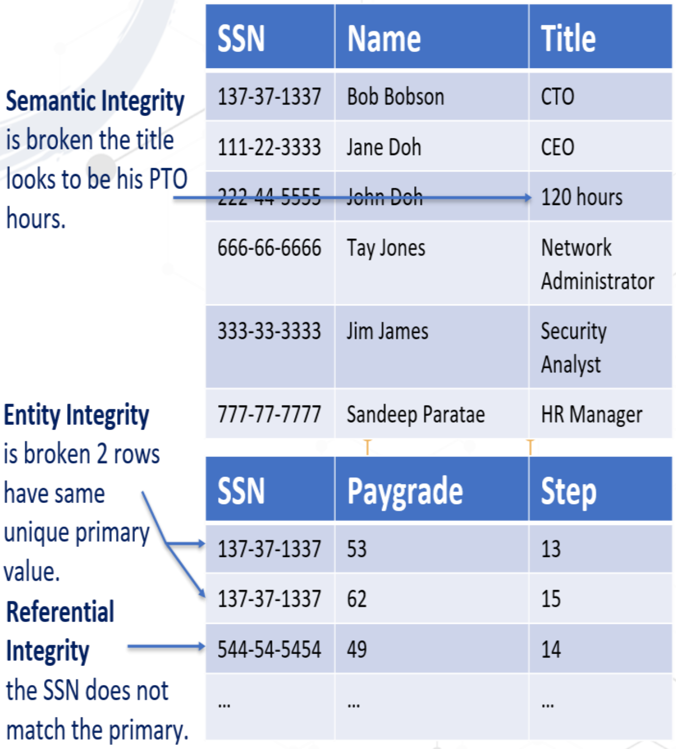
- **Relational Model:**
  - Organizes data into one or more tables (or relations) of columns and rows, with a unique key identifying each row.
  - Rows are also called records or tuples.
  - Generally, each table/relation represents one entity type.
  - The rows represent instances of that type of entity and the columns representing values attributed to that instance.
  - **Foreign Key:**
    - They are in relational databases the matching primary key of a parent database table.
    - It is always the primary key in the local DB.
    - The SSN is Primary key in the Paygrade/scale table, but Foreign key in the Name one, seen from the Paygrade/scale table.



SSN	Name	Title
137-37-1337	Bob Bobson	CTO
111-22-3333	Jane Doh	CEO
222-44-5555	John Doh	CFO
666-66-6666	Tay Jones	Network Administrator
333-33-3333	Jim James	Security Analyst
777-77-7777	Sandeep Paratae	HR Manager

SSN	Paygrade	Step
137-37-1337	53	13
111-22-3333	62	15
222-44-5555	49	14
…	…	…

- **Integrity:**
  - **Referential integrity:**
    - When every foreign key in a secondary table matches a primary key in the parent table.
    - It is broken if not all foreign keys match the primary key.
  - **Semantic integrity:**
    - Each attribute value is consistent with the attribute data type.
  - **Entity integrity:**
    - Each tuple (row) has a unique primary value that is not null.



Semantic Integrity is broken the title looks to be his PTO hours.

Entity Integrity is broken 2 rows have same unique primary value.

Referential Integrity the SSN does not match the primary.

SSN	Name	Title
137-37-1337	Bob Bobson	CTO
111-22-3333	Jane Doh	CEO
222-44-5555	John Doh	120 hours
666-66-6666	Tay Jones	Network Administrator
333-33-3333	Jim James	Security Analyst
777-77-7777	Sandeep Paratae	HR Manager

SSN	Paygrade	Step
137-37-1337	53	13
137-37-1337	62	15
544-54-5454	49	14
…	…	…

- **User-Defined Integrity:**
  - A set of rules specified by a user, which do not belong to the entity, domain and referential integrity categories.
  - If a database supports these features, it is the responsibility of the database to ensure data integrity as well as the consistency model for the data storage and retrieval.
  - If a database does not support these features it is the responsibility of the applications to ensure data integrity while the database supports the consistency model for the data storage and retrieval.
- Having a single, well controlled, and well defined data-integrity system increases:
  - **Stability**: One centralized system performs all data integrity operations
  - **Performance**: All data integrity operations are performed in the same tier as the consistency model.
  - **Re-usability**: All applications benefit from a single centralized data integrity system.
  - **Maintainability**: One centralized system for all data integrity administration.
- Modern databases support these features, and it has become the de facto responsibility of the database to ensure data integrity.
- If our databases are older we can use companies or database systems, who offer products and services to migrate legacy systems to modern databases.
- Databases normally run multiple threads simultaneously and they are all capable of altering data.
- When two threads try to change the same record, the DBBMS will attempt to commit the update.
- If the commit is unsuccessful, the DBMSs can do rollbacks/aborts and restore from a save point.
- A database journal is a log of all database transactions.
- If a database become corrupted, the database can be reverted to a back-up copy, and then transactions are replayed from the journal, restoring database integrity.

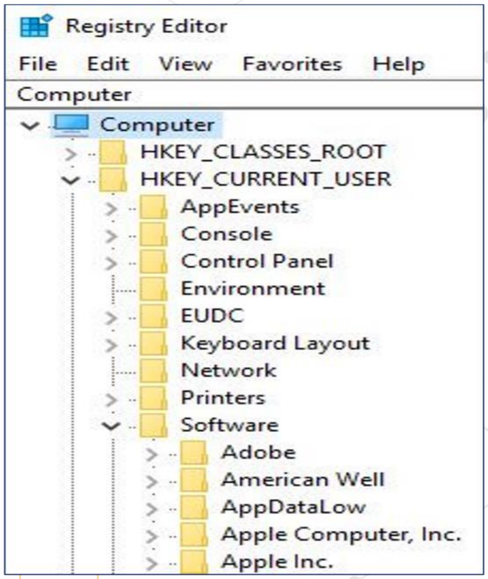
- **Database Normalization:**
  - Used to clean up the data in a database table to make it logically concise, organized, and consistent.
  - Removes redundant data, and improves the integrity and availability of the database.
  - Normalization has three forms (rules):
    - **First Normal Form:** Divides the base data into tables, primary key is assigned to most or all tables.
    - **Second Normal Form:** Move data that is partially dependent on the primary key to another table.
    - **Third normal Form:** Remove data that is not dependent on the primary key.

- The major benefits of using normalization include:
  - Greater overall database organization
  - Reduction of redundant data
  - Data consistency within the database
  - A much more flexible database design
  - A better handle on database security

- **Database Views:**
  - Database tables may be queried, what we see when we query them is called a database view.
  - They can give users a view of the parts of the database they are allowed to access.
  - For a normal employee this could be their own employee data, where HR can access all employee's data. Remember the need to know principle, even if you have the access that doesn't mean you are allowed to access it.
- **Data Dictionary:**
  - Contains a description of the database tables (metadata).
  - It has the database view information, information about authorized database administrators, user accounts names and privileges, auditing information, database schema,...
  - **Database Schema:**
    - Describes the attributes and values of the database tables.
    - Names should only contain letters, in the US SSN's should only contain 9 numbers,...

- **Database Query Language:**
  - Allow the creation, modification and deletion of database tables, the read/write access for those tables,...
  - Database query languages have at least two subsets of commands:
  - **Data Definition Language (DDL):**
    - A standard for commands that define the different structures in a database.
    - Creates, modifies, and removes database objects such as tables, indexes, and users.
    - Common DDL statements are CREATE, ALTER, and DROP.
  - **Data Manipulation Language (DML).**
    - Used for selecting, inserting, deleting and updating data in a database.
    - Common DML statements are SELECT, DELETE, INSERT, UPDATE.
  - SQL or a SQL derivatives are by far the most common query languages.

- **Hierarchical Databases**
  - Use a tree-like structure for how data is organized.
  - The data is stored as records which are connected to one another through links.
  - A record is a collection of fields, with each field containing only one value.
  - The entity type of a record defines which fields the record contains.



Windows Registry is an example of a hierarchical database.

- **Object-Oriented Databases (Object Database Management Systems):**
  - Object databases store objects rather than data such as integers, strings or real numbers.
  - Objects are used in object oriented languages such as Smalltalk, C++, Java,...
  - Objects, in an object-oriented database, reference the ability to develop a product, then define and name it.
  - The object can then be referenced, or called later, as a unit without having to go into its complexities.
  - Objects basically consist of the following:
  - **Attributes:**
    - Data which defines the characteristics of an object.
    - This data may be simple such as integers, strings, and real numbers or it may be a reference to a complex object.
  - **Methods:**
    - Defines the behavior of an object and are what was formerly called procedures or functions.
    - Objects contain both executable code and data.
  - **Classes:**
    - Define the data and methods the object will contain, they are the template for the object.
    - Does not itself contain data or methods but defines the data and methods contained in the object.

- **We have covered these before, but it makes sense to revisit them now:**
  - **Database Shadowing:**
    - Exact real time copy of the database or files to another location.
    - It can be another disk in the same server, but best practices dictates another geographical location, often on a different media.
  - **Electronic Vaulting** (E-vaulting):
    - Using a remote backup service, backups are sent off-site electronically at a certain interval or when files change.
  - **Remote Journaling:**
    - Sends transaction log files to a remote location, not the files themselves. The transactions can be rebuilt from the logs if we lose the original files.

- **Coupling:**
  - The degree of interdependence between software modules, a measure of how closely connected two routines or modules are.
- **Cohesion:**
  - Refers to the degree to which the elements inside a module belong together.
  - Measures the strength of relationship between pieces of functionality within a given module.
  - In highly cohesive systems functionality is strongly related.
- Coupling is usually contrasted with cohesion.
- Low coupling often correlates with high cohesion, and vice versa.
- Low coupling is often a sign of a well-structured computer system and a good design, and when combined with high cohesion, supports the general goals of high readability and maintainability**.**
- **ORB (Object Request Broker):**
  - Middleware which allows program calls to be made from one computer to another via a network, providing location transparency through remote procedure calls.
  - ORBs promote interoperability of distributed object systems, enabling such systems to be built by piecing together objects from different vendors, while different parts communicate with each other via the ORB.
  - Common object brokers included .NET remoting, COM, DCOM, and CORBA.
    - **COM (Component Object Model):**
      - A language-neutral way of implementing objects that can be used in environments different from the one in which they were created, even across machine boundaries.
      - It is used to enable inter-process communication object creation in a large range of programming languages.

    - **DCOM (Distributed COM):**
        - The networked sequel to COM which adds to support communication among objects on different computers—on a LAN, a WAN, or even the Internet.
        - The application can be distributed at locations that make the most sense to your customer and to the application itself.
        - DCOM includes Object Linking and Embedding (OLE), a way to link documents to other documents.
        - Both COM and DCOM are slowly being replaced by Microsoft.NET, which can interoperate with DCOM, but offers more advanced functionality than COM and DCOM.
    - **CORBA (Common Object Request Broker Architecture):**
        - Open vendor neutral ORB standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms.
        - Enables collaboration between systems on different operating systems, programming languages, and computing hardware.
        - CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented.

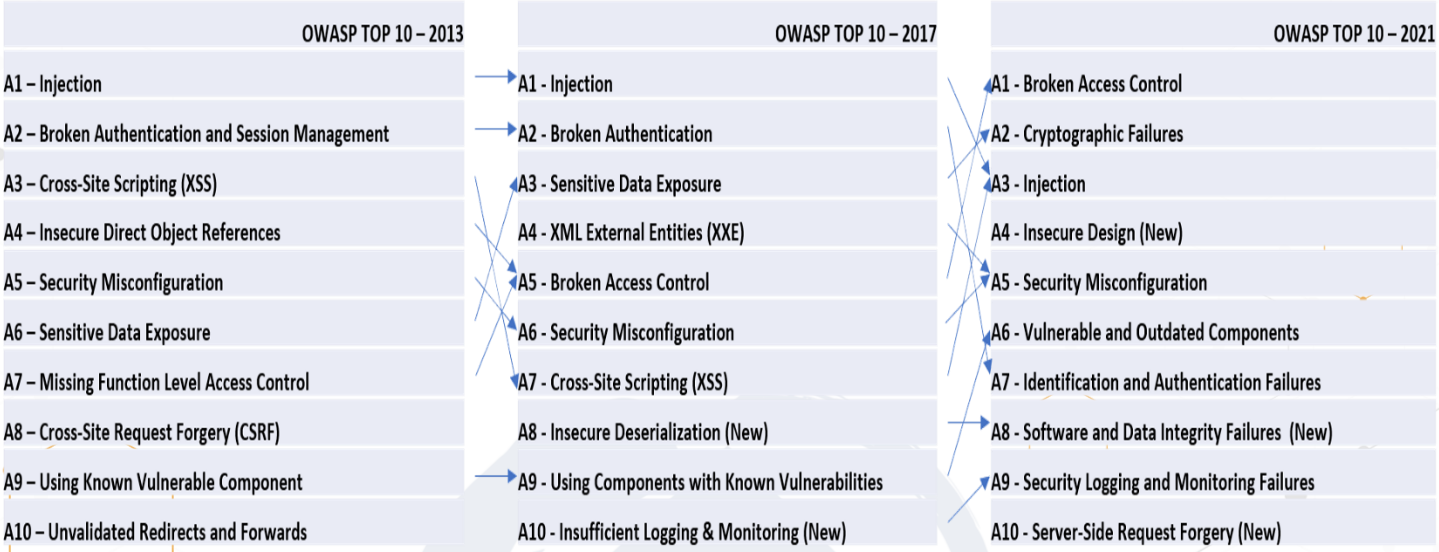
- **OOAD (Object-Oriented Analysis and Design):**
    - Iteration after iteration, the outputs of OOAD activities, analysis models for OOA and design models for OOD respectively, will be refined and evolve continuously driven by key factors like risks and business value.
    - **OOA (Object-Oriented Analysis):**
        - Creates a model of the system's functional requirements that is independent of implementation constraints.
        - Organizes requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with.
        - The primary tasks are:
            - Find the objects, Organize the objects, Describe how the objects interact, Define the behavior of the objects, Define the internals of the objects.
    - **OOD (Object-Oriented Design):**
        - The developer applies the constraints to the conceptual model produced in object-oriented analysis.
        - Such constraints could include the hardware and software platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time.
        - Concepts in the analysis model which is technology independent, are mapped onto implementing classes and interfaces resulting in a model of the how the system is to be built on specific technologies.

        - Important topics during OOD also include the design of software architectures by applying architectural patterns and design patterns with object-oriented design principles.
    - **OOM (Object-Oriented Modeling):**
        - Common approach to modeling applications, systems, and business domains by using the object-oriented paradigm throughout the entire development life cycles.
        - Heavily used by both OOA and OOD activities in modern software engineering.

- **The ACID model (Atomicity, Consistency, Isolation, and Durability):**
    - The ACID model is based on 4 parts:

        - **Atomicity**:
            - All or nothing, if any part of the transaction fails, the entire transaction fails.
        - **Consistency**:
            - The database must be consistent with the rules, before and after the transaction.
        - **Isolation**:
            - One transaction must be completed before another transaction can modify the same data.
        - **Durability**:
            - Once transactions are committed to the database they must be preserved.

- **Software Vulnerabilities and Attacks:**
  - **OWASP (Open Web Application Security Project):**
    - Top 10 of the most common web security issues.



OWASP TOP 10 – 2013	OWASP TOP 10 – 2017	OWASP TOP 10 – 2021
A1 – Injection	A1 - Injection	A1 - Broken Access Control
A2 – Broken Authentication and Session Management	A2 - Broken Authentication	A2 - Cryptographic Failures
A3 – Cross-Site Scripting (XSS)	A3 - Sensitive Data Exposure	A3 - Injection
A4 – Insecure Direct Object References	A4 - XML External Entities (XXE)	A4 - Insecure Design (New)
A5 – Security Misconfiguration	A5 - Broken Access Control	A5 - Security Misconfiguration
A6 – Sensitive Data Exposure	A6 - Security Misconfiguration	A6 - Vulnerable and Outdated Components
A7 – Missing Function Level Access Control	A7 - Cross-Site Scripting (XSS)	A7 - Identification and Authentication Failures
A8 – Cross-Site Request Forgery (CSRF)	A8 - Insecure Deserialization (New)	A8 - Software and Data Integrity Failures (New)
A9 – Using Known Vulnerable Component	A9 - Using Components with Known Vulnerabilities	A9 - Security Logging and Monitoring Failures
A10 – Unvalidated Redirects and Forwards	A10 - Insufficient Logging & Monitoring (New)	A10 - Server-Side Request Forgery (New)

https://owasp.org/www-project-top-ten/

- **OWASP:**
  - **A01:2021 - Broken Access Control:**
    - It is not implemented consistently across an entire application.
    - It can be done correctly in one location but incorrectly in another.
    - We need a centralized access control mechanism, and we write the tricky logic once and reuse it everywhere.
    - This is essential both for writing the code correctly and for making it easy to audit later.
    - Many access control schemes were not deliberately designed but have simply evolved along with the website.
    - Inconsistent access control rules are often inserted in various locations all over the code, making it near impossible to manage.
    - One especially dangerous type of access control vulnerability arises from web-accessible administrative interfaces, frequently used to allow site administrators to efficiently manage users, data, and content on their site.

    - **What can we do?**
      We can deny by default, limit user rights, use role-based access control, strong passwords, MFA, log/act on access control failures, proper user and session management,...
  - https://owasp.org/Top10/A01_2021-Broken_Access_Control/

- **A02:2021 - Cryptographic Failures:**
  - Sites are HTTP rather than HTTPS.
  - Data is sent in cleartext.
  - Backups, data at rest and data in transit are not encrypted (stored/transmitted in plain text).
  - Using older, weaker, and deprecated encryption algorithms.
  - Using depreciated hash functions.
  - Not monitoring if data is being exfiltrated.
  - Improper use of initialization vectors.

  - **What can we do?**
    We ensure we do not use depreciated encryption, data is identified and protected properly, no clear-text, proper implementation of up-to-date encryption/protocols/keys, no caching for responses with sensitive data, only store sensitive data as long as required,…
- https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

- **A03:2021 – Injection:**
  - Can be any code injected into user forms. Often seen is SQL/NoSQL/OS command/LDAP.
  - Attackers can do this because our software does not use:
    - Strong enough input validation and data type limitations input fields.
    - Input length limitations.
  - **CGI (Common Gateway Interface):**
    - Standard protocol for web servers to execute programs running on a server that generates web pages dynamically. We use the interface to ensure only proper input makes it to the database.
    - The CGI separates the untrusted (user) from the trusted (database).

  - **What can we do?**
    - The fix is to do just that, we only allow users to input appropriate data into the fields, only letters in names, numbers in phone number, have dropdowns for country and state (if applicable), we limit how many characters people can use per cell, use secure APIs,...
    - Separating the data from the web application logic.
    - Implement settings and/or restrictions to limit data exposure in case of successful injection attacks.
- https://owasp.org/Top10/A03_2021-Injection/

- **A04:2021 - Insecure Design:**
  - When we design our web applications, we need to design them securely.
  - This does not have to be design flaws, it can also be anything that is not secure, any weakness that an attacker could exploit.
  - Not to be confused insecure implementation.
    - We can have a securely designed app and still implement it insecurely.
    - However, we can't fix an insecure design with a flawless implementation.
  - **What can we do?**
    - We have our software developers use secure design patterns and reference architectures to build applications.
    - Our organization should have libraries with references and patterns.
    - Before finalizing our application design, we use a red team to do threat modeling and penetration testing.
- https://owasp.org/Top10/A04_2021-Insecure_Design/

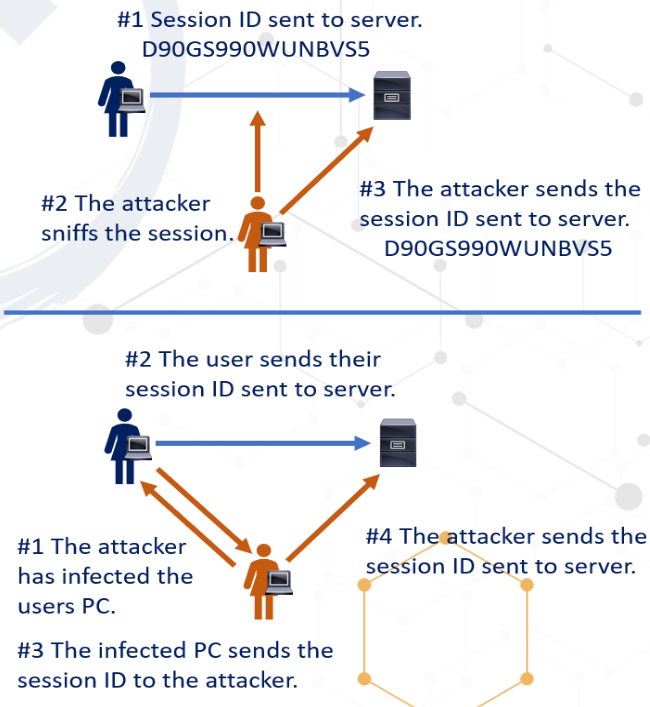
- **A05:2021 - Security Misconfiguration:**
  - Databases configured wrong.
  - Not removing out-of-the-box default access and settings.
  - Keeping default usernames and passwords.
  - VM, OS, webserver, DBMS, applications,… are not patched and up to date.
  - Unnecessary features are enabled or installed. This could be open ports, services, pages, accounts, privileges,…
  - With so much being cloud now, it is only natural that misconfiguration is more prevalent, so many more options admins can disable.

  - **What can we do?**
  - It is pretty simple; server hardening, proper patching, do not disable security features unless we are completely clear on why and we have done proper risk analysis.
- https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

- **A06:2021 - Vulnerable and Outdated Components:**
  - Vulnerable components can be both client and server-side (OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, libraries,…).
  - Developers use deprecated code or objects that are known to be unsecure.
    - Mostly happens because developers are used to the old code or the library, they could be uncertain about new code, or they are afraid to break anything.

  - **What can we do?**
  - Proper patch management, scan for vulnerabilities, make sure we don't use deprecated code, keep a continuous inventory of both server-side and client-side components and their dependencies, delete unused programs and features.
- https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

- **A07:2021 - Identification and Authentication Failures:**
  - Sessions do not expire or take too long to expire.
  - Session IDs are predictable or part of the URL; 001, 002, 003, 004,…
  - Tokens, Session IDs, Passwords,… are kept in plaintext or are poorly protected (poor encryption and hashing).
  - Weak/default passwords and knowledge based password recovery.


#1 Session ID sent to server.
D90GS990WUNBVS5
#2 The attacker sniffs the session.
#3 The attacker sends the session ID sent to server.
D90GS990WUNBVS5
#2 The user sends their session ID sent to server.
#1 The attacker has infected the users PC.
#4 The attacker sends the session ID sent to server.
#3 The infected PC sends the session ID to the attacker.

  - **What can we do?**
  - MFA, sessions expires, non-predictable sessions, no plain-text anywhere, no session ID in URL, proper secure encryption, no default/weak passwords, log login failures, alert admins when detecting brute force, credential stuffing, and any other attacks.
- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

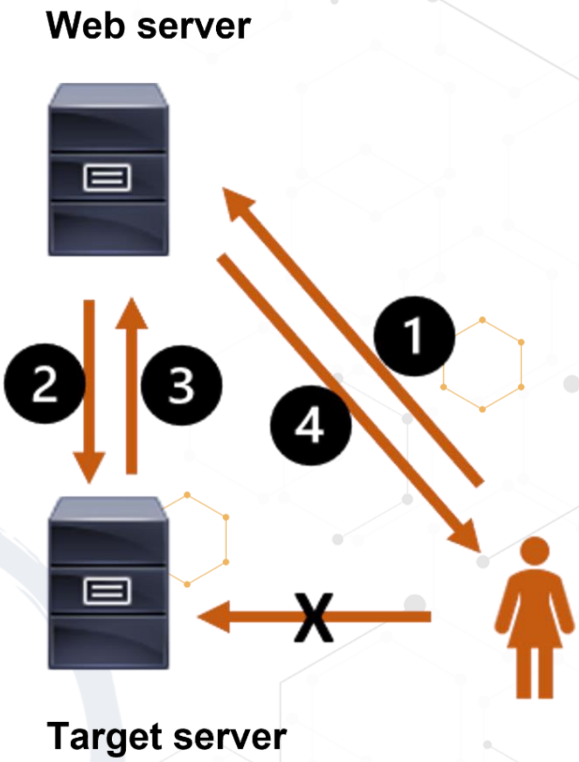
- **A08:2021 - Software and Data Integrity Failures:**
  - When our applications use code, plugins, libraries, or modules from untrusted sources.
  - Insecure CI/CD pipelines or unverified updates.
  - Software with automatic updates without enough integrity checks.

  - **What can we do?**
  - Use digital signatures/hashes to verify the code/data is from the right source and is unaltered.
  - Make sure libraries and dependencies are using trusted repositories.
  - Deploy software supply chain tools to make sure components do not contain known vulnerabilities.
  - Ensure our CI/CD pipeline has proper segregation, configuration, and access control.
    - This should ensure the integrity of the code throughout the build and deploy processes.
- https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/

- **A09:2021 - Security Logging and Monitoring Failures:**
  - When our intrusion monitoring and reporting system fail to catch and report signs of intrusion.
  - Result of poor configuration, low thresholds, or logs saved just locally.
  - Attacks go unnoticed if we do not act on appropriate logs or alerts.

  - **What can we do?**
  - Implement proper monitoring and logging, ensure we log/report all failed login attempts and server-side validations.
  - Logs are generated in a format that our log management system can easily use, logs are kept long enough.
  - Logs are kept secure and protected against injection or any other type of attack.
  - Audit trails on high-value transactions.
  - Have a proper incident response and recovery plan.
- https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/

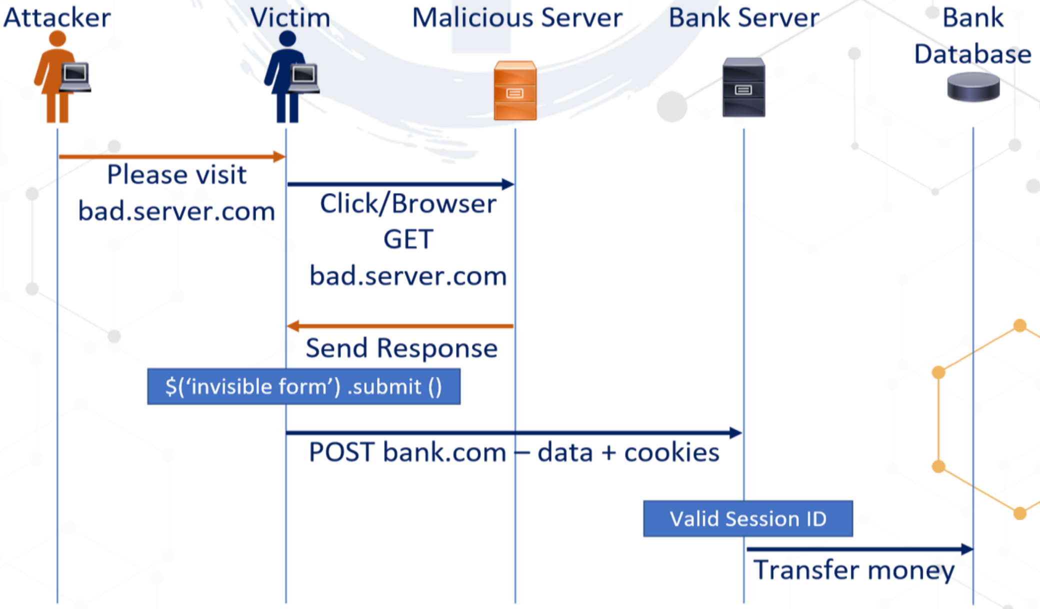
- **A10:2021 - Server-Side Request Forgery:**
  - Web applications usually trigger requests between HTTP servers, to fetch remote resources, such as software updates, or to import metadata from a URL or another web application.
  - Usually benign, but if not implemented correctly, they can make a server vulnerable to SSRF.
  - Normally an attacker can't access an internal server because it would be blocked by the firewall. To get around that the attacker can exploit an SSRF vulnerability to launch their attack using a vulnerable web server.
  - The attacker changes a parameter value in the vulnerable web application to create or control requests from the vulnerable server.



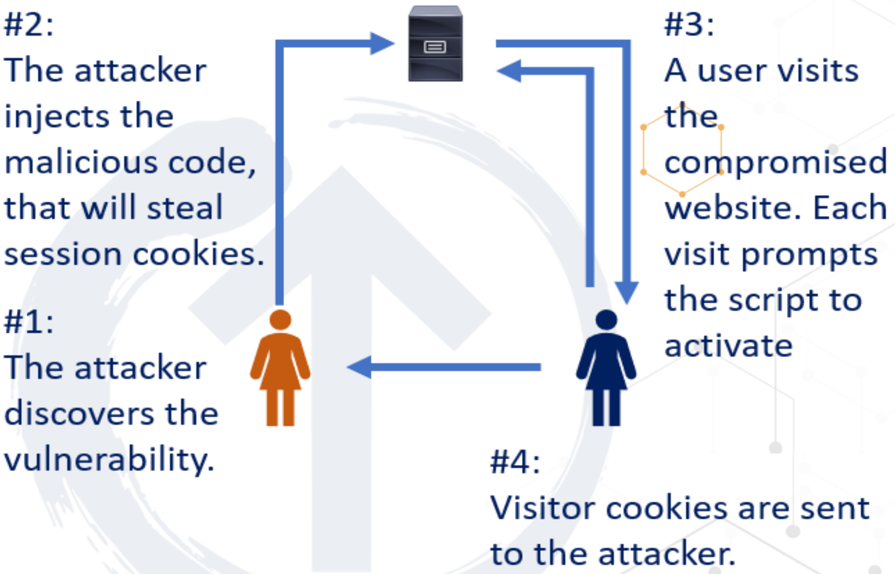
  - **What can we do?**
    - **Network layer:** Segment remote resource access functionality in separate networks. Enforce "deny by default" to block all non-essential intranet traffic.
    - **Application layer:** Sanitize and validate all client-supplied input data.
    - Enforce the URL schema, port, and destination with a positive allow list.
    - Do not send raw responses to clients. Disable HTTP redirections.
- https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/

- **Insufficient Detection and Response:**
  - Not detecting we have been compromised due to lack of controls and/or detection applications.
  - Not performing our due diligence and due care on our applications, systems, and our response to compromise.
  - Not responding in a proper way to compromise, not informing anyone, informing too late, or just ignoring the incident (at best plugging the leak).
  - We need to not just protect against this attack, but future similar attacks, patch software, and applications, close ports.

- **Unvalidated Redirects and Forwarding:**
  - Not confirming URLs forward and redirecting us to the right page.
  - Mitigated with user awareness and spider/crawl our site to see if it generates any redirects (HTTP response codes 300-307, typically 302).
- **Cross-Site Request Forgery (CSRF):**
  - Stolen session IDs or tokens.
  - Often phishing.
  - Passwords/Username saved in cookies.
  - Saved site passwords, not logging off when done, using the same browser for sensitive and non-sensitive information.
  - Current browsers do mitigate some of this, they should use unique session-specific tokens (random or pseudo-random), and validate session tokens are not replayed.


Attacker
Victim
Malicious Server
Bank Server
Bank Database
Please visit bad.server.com
Click/Browser GET bad.server.com
Send Response
$('invisible form') .submit ()
POST bank.com – data + cookies
Valid Session ID
Transfer money

- **Cross-Site Scripting (XSS):**
    - Attackers inject client-side scripts into web pages viewed by other users.
    - The vulnerability may be used by attackers to bypass access controls such as the same-origin policy.
    - To prevent XSS, we can use proper input validation and data typing.
    - Set our server to redirect invalid requests, detect a simultaneous login from two different IP addresses and invalidate the sessions, require users to enter their passwords again before changing their registration information, and set cookie with HttpOnly flag to prevent access from JavaScript.


#2:
The attacker injects the malicious code, that will steal session cookies.
#3:
A user visits the compromised website. Each visit prompts the script to activate
#1:
The attacker discovers the vulnerability.
#4:
Visitor cookies are sent to the attacker.

- **Buffer Overflow (Buffer Overrun):**
    - An anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations, happen from improper coding when a programmer fails to perform bounds checking.
    - Buffers are areas of memory set aside to hold data, often while moving it from one section of a program to another, or between programs.
    - Buffer overflows can often be triggered by malformed inputs, if one assumes all inputs will be smaller than a certain size and the buffer is created to be that size, if an anomalous transaction produces more data it could cause it to write past the end of the buffer.
    - If this overwrites adjacent data or executable code, this may result in erratic program behavior, including memory access errors, incorrect results, and crashes.
    - By sending in data designed to cause a buffer overflow, it is possible to write into areas known to hold executable code, and replace it with malicious code.

- **Race Condition (Race Hazard):**
  - Two or more programs may collide in their attempts to modify or access a file.
  - This can be an attacker with access, altering files which can then result in data corruption or privilege escalation.
  - **TOCTOU (Time of Check to Time of Use):**
    - A software bug caused by changes in a system between the checking of a condition (such as a security credential) and the use of the results of that check.

- **Privilege Escalation:**
  - Exploiting a bug, design flaw or configuration oversight in an OS or application to gain access to resources that are normally protected from an application or user.
  - Attacker often use this to elevate the user account they have gained access to, in order to get administrator access.
  - The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions.

- **Backdoors:**
  - Often installed by attackers during an attack to allow them access to the systems after the initial attack is over, to exfiltrating data over time or to come back and compromise other systems.
  - Bypassing normal authentication or encryption in a computer system, a product, or an embedded device,...
  - Backdoors are often used for securing remote access to a computer, or obtaining access to plaintext in cryptographic systems.

- **Ethical disclosure:**
  - What do you do when you discover a vulnerability? We covered some of this in the white, gray, black hat hacker section.
  - **Full Disclosure:** Tell everyone, make it public, assuming attackers already know and are using it.
  - **Responsible/Partial Disclosure:** Telling the vendor, they have time to develop a patch and then disclose it.
    - If they do nothing we can revert to the full disclosure forcing them to act.
  - **No Disclosure:** Attackers finding a vulnerability would try to exploit it and keep it secret as long as possible.

- **Security Orchestration, Automation, and Response (SOAR):**
  - A software solution that uses AI to allows us to respond to some security incidents automatically.
  - SOAR vs. SIEM: Very similar, both detect and alert on security events, but using AI, SOAR will also react to some security events.
    - SIEMs often generate more alerts than a SOC team can handle, SOAR can help reduce the number of alerts and make workflows more manageable.
  - SOAR combines all the comprehensive data we gather, has case management, standardization, workflows, and analytics, and it can integrate with many of our other solutions (Vulnerability Management (VM), IT Service Management (ITSM), Threat Intelligence, …).
  - All this can help our organization implement a detailed defense-in-depth solution.

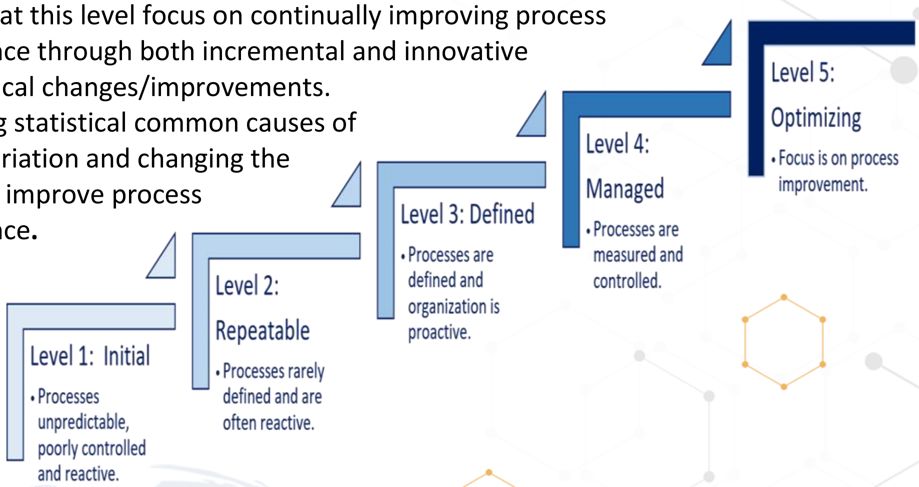
- **Operation and Maintenance:**
  - Once our finished software/project is handed off to operations, there will still be some maintenance tasks our organization needs to perform.
  - Our environment and the requirements for our applications is never static.
  - We need a solid support team in place to make sure the software functions as required, that any required changes are implemented using proper change management, and that all this is done with security in mind.

- **Integrated Development Environment (IDE):**
  - Applications that help in the development of other applications.
  - They are designed to contain all programming tasks in a single application, having a single central interface with all the tools the developer needs, including:
    - **The Code editor:** For writing and editing source code, these editors are different from text editors, they are designed to either simplify or enhance the process of writing and editing the code.
    - **Compiler:** The compilers change our source code, which is written in a human-readable language, into a form that computers can execute.
    - **Debugger:** Debuggers are used during the testing phase and can help our developers debug their code.
    - **Build automation tools:** Tools to help automate common dev tasks to save time.
    - On top of this some IDEs may also include:
      - **Class browser:** Used to reference and study the properties of an object-oriented class hierarchy.
      - **Object browser:** Used to inspect objects present in a running application program.
      - **Class hierarchy diagram:** Helps devs to visualize the structures of object-oriented programming code.

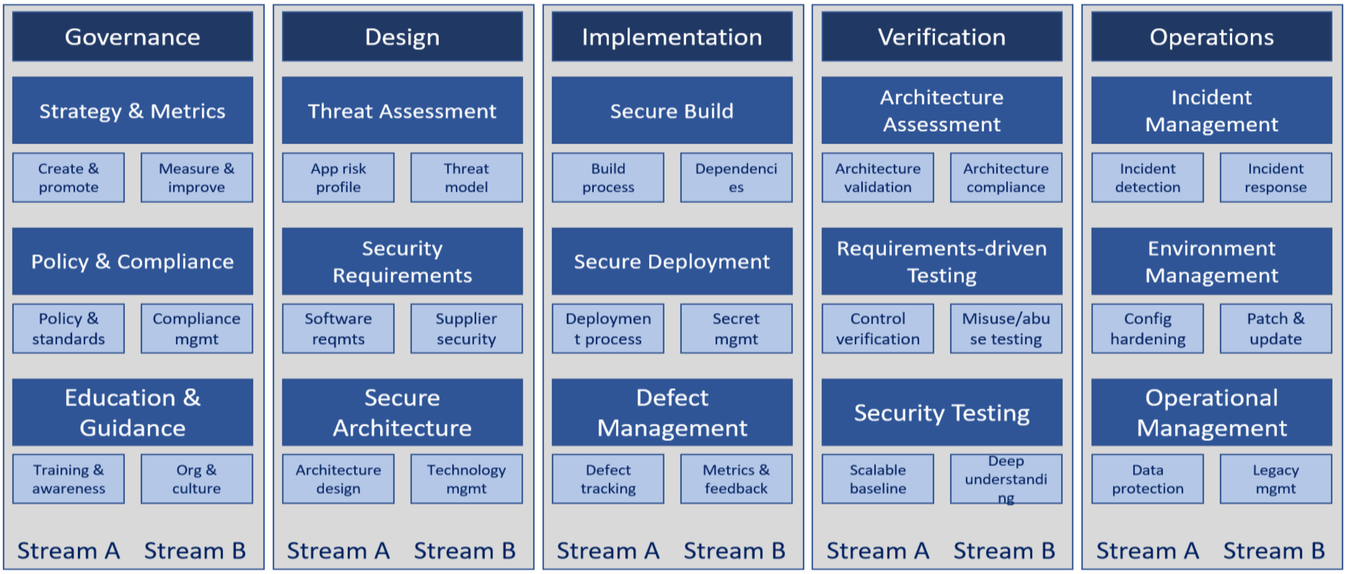
- **Runtime:**
  - Runtime is the amount of time when a program is running. Starting when a program is executed/started and stopping with the program terminated/closed.
  - The term, runtime, is most often used in software development. Commonly used with "runtime error," an error that occurs while a program is running. This error is used to differentiate from other types of errors, like syntax errors and compilation errors, which happen before a program is run.

- **CMM (Capability Maturity Model):**
  - The maturity relates to the degree of formality and optimization of processes, from ad hoc practices, to formally defined repeatable steps, to managed result metrics, to active optimization of the processes.
  - There are five levels defined in the model and, which describes where an organization is, it also has practical steps to how to mature the organization to get to the next level.
  - **Level 1: Initial**
    - Processes at this level are normally undocumented and in a state of dynamic change, tending to be driven in an ad hoc, uncontrolled and reactive manner by users or events.
    - This provides a chaotic or unstable environment for the processes.
  - **Level 2: Repeatable**
    - This level of maturity that some processes are repeatable, possibly with consistent results.
    - Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.
  - **Level 3: Defined**
    - This level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time.
    - These standard processes are in place.
    - The processes may not have been systematically or repeatedly utilized enough for the users to become competent or the process to be validated in a range of situations.
  - **Level 4: Managed (Capable)**
    - Processes at this level uses process metrics, effective achievement of the process objectives can be evidenced across a range of operational conditions.
    - The suitability of the process in multiple environments has been tested and the process refined and adapted.
    - Process users have experienced the process in multiple and varied conditions, and are able to demonstrate competence.

- **Level 5: Optimizing**
  - Processes at this level focus on continually improving process performance through both incremental and innovative technological changes/improvements.
  - Addressing statistical common causes of process variation and changing the process to improve process performance**.**


Level 1: Initial
• Processes unpredictable, poorly controlled and reactive.
Level 2: Repeatable
• Processes rarely defined and are often reactive.
Level 3: Defined
• Processes are defined and organization is proactive.
Level 4: Managed
• Processes are measured and controlled.
Level 5: Optimizing
• Focus is on process improvement.

Software Assurance Maturity Model (SAMM): https://owaspsamm.org/model/



Governance		Design		Implementation		Verification		Operations	
Strategy & Metrics		Threat Assessment		Secure Build		Architecture Assessment		Incident Management	
Create & promote	Measure & improve	App risk profile	Threat model	Build process	Dependencies	Architecture validation	Architecture compliance	Incident detection	Incident response
Policy & Compliance		Security Requirements		Secure Deployment		Requirements-driven Testing		Environment Management	
Policy & standards	Compliance mgmt	Software reqmts	Supplier security	Deployment process	Secret mgmt	Control verification	Misuse/abuse testing	Config hardening	Patch & update
Education & Guidance		Secure Architecture		Defect Management		Security Testing		Operational Management	
Training & awareness	Org & culture	Architecture design	Technology mgmt	Defect tracking	Metrics & feedback	Scalable baseline	Deep understanding	Data protection	Legacy mgmt
Stream A	Stream B	Stream A	Stream B	Stream A	Stream B	Stream A	Stream B	Stream A	Stream B

Software Assurance Maturity Model (SAMM)

- The Software Assurance Maturity Model (SAMM) is an open framework that provides an effective and measurable way for all types of organizations to analyze and improve their software security posture.
- **The resources provided by SAMM aid in:**
  - Evaluating an organization's existing software security practices.
  - Building a balanced software security assurance program in well-defined iterations.
  - Demonstrating concrete improvements to a security assurance program.
  - Defining and measuring security-related activities throughout an organization.

- **SAMM principles**
    - An organization's behavior changes slowly over time.
    - Changes must be iterative while working toward long-term goals.
    - There is no single recipe that works for all organizations.
    - A solution must enable risk-based choices tailored to the organization.
    - Guidance related to security activities must be prescriptive.
    - A solution must provide enough details for non-security-people.
    - Overall, it must be simple, well-defined, and measurable.
- https://owasp.org/www-project-samm/

- **Acceptance Testing:**
    - There are many different testing types we use throughout the development lifecycle.
    - At the end of development we also use acceptance testing, we need to test it to ensure it does what it is supposed to and it is robust and secure.

    - **The User Acceptance Test:**
        - Is the software functional for the users who will be using it? It is tested by the users and application managers.

    - **Operational Acceptance Testing:**
        - Does the software and all of the components it interacts with ready requirements for operation.
        - Tested by system administrators are the backups in place, do we have a DR plan, how do we handle patching, is it checked for vulnerabilities,...?

    - **Contract Acceptance Testing:**
        - Does the software fulfil the contract specifications? The what/where/how of the acceptance is defined in the contract.

    - **Compliance Acceptance Testing:**
        - Is the software compliant with the rules, regulations and laws of our industry?

    - **Compatibility/Production Testing:**
        - Does the software interface as expected with other applications or systems?
        - Does the software perform as expected in our production environment vs. the development environment?

- **Buying software from other companies:**
  - When we buy software from vendors either COTS (Commercial Off The Shelf) or custom built software we need to ensure it is as secure as we need it to be.
  - Vendors claims of security posture should until proven be seen as marketing claims.
  - We need to do our due care and due diligence, as well as use outside council if needed.
  - Many organizations deal with C-level executives going to conferences and buying software that the organization may not want or need.
  - Software development and procurement as well as any other project should be carefully scoped, planned be based on a clear analysis of what the business needs and wants.

  - **COTS (Commercial Off-the-Shelf) Software:**
    - When buying COTS software we can, depending on how widely the software is used, look at reviews, talk to current customers and users to get a clearer understanding of the software capabilities and security.
    - Software roadmaps are nice, but only buy the software for what it can actually do now, not what it can maybe do in the future.
    - We can use a clear RTM (requirements traceability matrix), requirements are divided into "Must have, nice to have and maybe should have".
    - We would then score the software candidates on the "Have's" and from that we should be able to see feasible candidates, other factors such as cost, maintenance also play a big part in the decision.
    - For large/expensive implementations it may also be possible for the vendor to provide references to talk to.
    - We would also look at how financially sound the vendor looks to be, if we spend $2,000,000 on software and the vendor goes out bankrupt in 3 months, we may have to spend another $2,000,000 all over again.

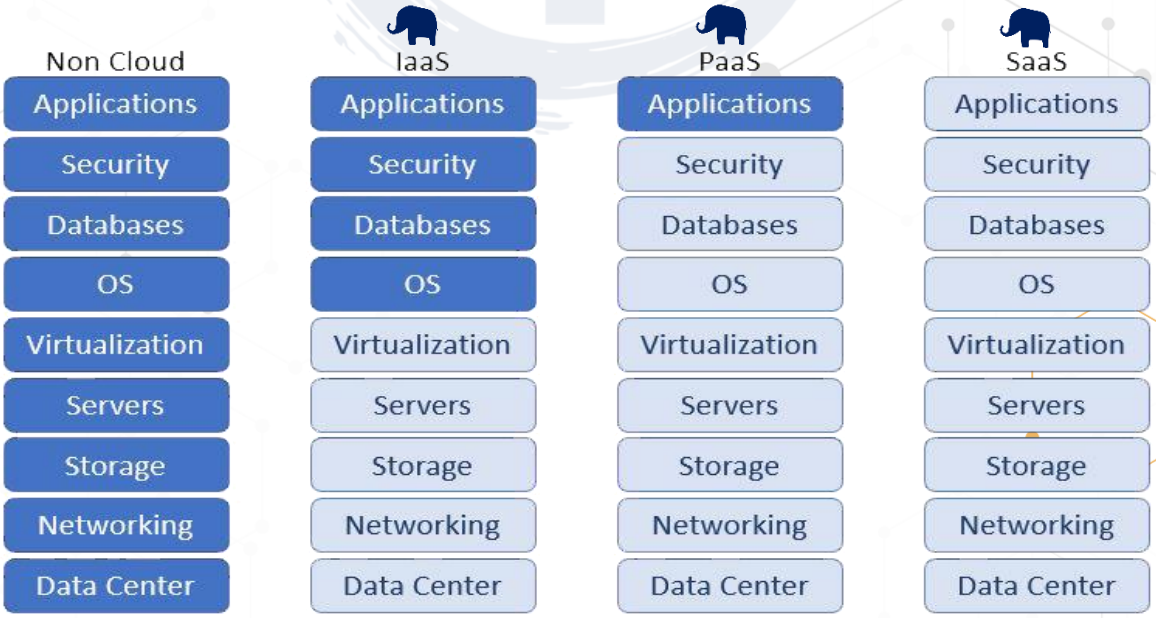
  - **Custom-Developed Third-Party Products:**
    - Having someone else develop the software we need is also an option.
    - This is higher cost than COTS software, but also far more customizable.
    - The same questions and then some should be asked:
    - How good are they? Have they done this before? How secure are they?,...
    - Do we own the code or do we rent it when it is done?
    - What happens if they go out of business?
    - Who will support it?
    - Do you have capable staff, that can support and tweak the software?
    - Is it secure or is it security through obscurity?
    - Many code shops are just that, only code shops, once the software is accepted it is your problem to do the day to day maintenance, they may contract for updates, but that is it.

Cloud Computing:

- Cloud Computing can be divided into 4 main types:
    - **Private Cloud Computing** - Organizations build and run their own cloud infrastructure (or they pay someone to do it for them).
    - **Public Cloud Computing** - Shared tenancy – A company builds massive infrastructures and rents it out to anyone who wants it. (Amazon AWS, Microsoft, Google, IBM).
    - **Hybrid Cloud Computing** – A mix of Private and Public Cloud Computing. An organization can choose to use Private Cloud for sensitive information and Public Cloud for non-sensitive data.
    - **Community Cloud Computing** – Only for use by a specific community of consumers from organizations that have shared concerns. (Mission, policy, security requirements, and/or compliance considerations.)

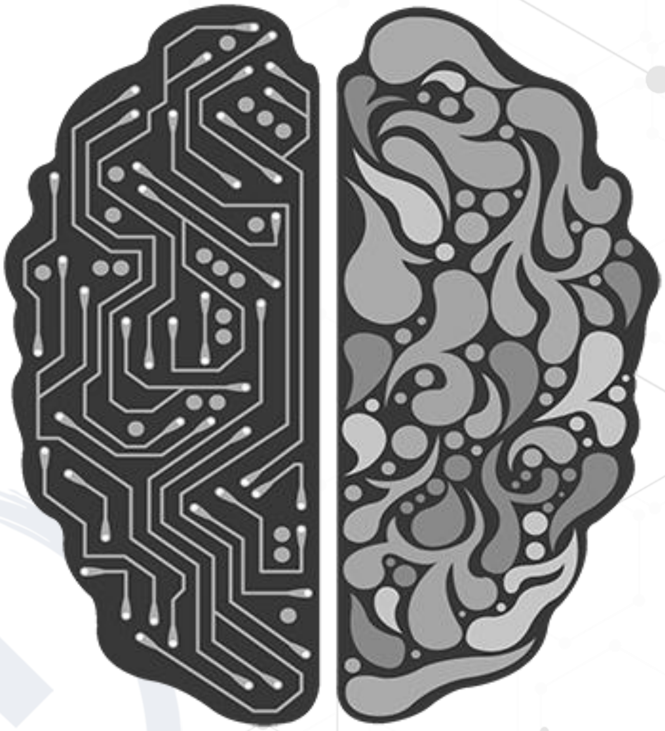
As with any other outsourcing, make sure you have the right to audit, pen test (clearly agreed upon criteria), conduct vulnerability assessment, and check that the vendor is compliant with your industry and the standards you adhere to.

- Platforms are normally offered as:
    - **IaaS (Infrastructure as a Service)** The vendor provides infrastructure up to the OS, the customer adds the OS and up.
    - **PaaS (Platform as a Service)** The vendor provides pre-configured OSs, then the customer adds all programs and applications.
    - **SaaS (Software as a Service)** The vendor provides the OS and applications/programs. Either the customer interacts with the software manually by entering data on the SaaS page, or data is automatically pushed from your other applications to the SaaS application (Gmail, Office 365, Dropbox, Payroll, …).



Non Cloud	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Security	Security	Security	Security
Databases	Databases	Databases	Databases
OS	OS	OS	OS
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking
Data Center	Data Center	Data Center	Data Center

AI (Artificial Intelligence):



- Intelligence exhibited by machines, rather than humans or other animals.
- What true AI is, is a topic of discussion, what was considered AI years ago we have achieved and when once goal is reached the AI definition is tweaked a little.
- From what we are seeing published we do in my mind not currently have true AI, but very highly simulated intelligence, that being said IBM and Google do seem to be getting a lot closer.
- It is also used when a machine mimics cognitive functions that humans associate with other human minds, such as learning and problem solving.
- AI currently defined as advice that perceives its environment and takes actions that maximize its chance of success at some goal, not through experience/programming, but through reasoning.

- **Expert Systems:**
  - A computer system that emulates the decision-making ability of a human expert.
  - Designed to solve complex problems by reasoning about knowledge, represented mainly as if–then rules rather than through conventional procedural code.
  - An expert system is divided into two subsystems:
    1. The knowledge base represents facts and rules.
    2. The inference engine applies the rules to the known facts to deduce new facts, and can also include explanation and debugging abilities.

- **ANN's (Artificial Neural Networks):**
  - Computing systems inspired by the biological neural networks that constitute animal brains, we make decisions based on 1000's of memories, stories, the situation and many other factors, the ANN tries to emulate that.
  - The systems learn and progressively improve their performance, to do tasks, generally without task-specific programming.
  - They can learn to identify images that contain geckos by analyzing example images that have been manually labeled as "gecko" or "no gecko" and using the analytic results to identify geckos in other images.
  - They are mostly used in areas that are difficult to express in a traditional computer algorithm using rule-based programming.
  - An ANN is based on a collection of connected units called artificial neurons.
  - Each connection (synapse) between neurons can transmit a signal to another neuron.
  - Typically, neurons are organized in layers, different layers may perform different transformations on their inputs.
  - Signals travel from the first input, to the last output layer, at times after traversing the layers multiple times.

- **GP (Genetic Programming):**
  - A technique where computer programs are encoded as a set of genes that are then modified (evolved) using an evolutionary algorithm often a GA (Genetic Algorithm).
  - The results are computer programs able to perform well in a predefined task.
  - The methods used to encode a computer program in an artificial chromosome and to evaluate its fitness with respect to the predefined task are central in the GP technique and still the subject of active research.
  - GP evolves computer programs, traditionally represented in memory as tree structures.
  - Trees can be easily evaluated in a recursive manner.
  - Every tree node has an operator function and every terminal node has an operand, making mathematical expressions easy to evolve and evaluate.
  - Traditionally GP favors the use of programming languages that naturally embody tree structures for example, Lisp or other functional programming languages.
  - The process is in its simple form like this:
    - Generate an initial population of random computer programs.
    - Execute each program in the population and assign it a fitness value according to how well it solves the problem.
    - Create a new population of computer programs.
    - Copy the best existing programs
    - Create new computer programs by mutation.
    - Create new computer programs by crossover.

- Genetic Algorithms and Genetic Programming have been used to program a Pac-Man playing program, robotic soccer teams, networked intrusion detection systems, and many others.

What we covered in the Eighth CBK Domain:

This chapter we covered how we design security into our software as we develop it:

- Security in the **software development lifecycle**.
- **Project Management methodologies**.
- DevOps and DevSecOps.
- Development environment security controls.
- How we assess the **effectiveness of our software security controls**.
- **OWASP top 10** and other software vulnerabilities.
- Software development **maturity models**.
- **3rd party software security.**
- **AI** (Artificial Intelligence).

- CBK 8 makes up 10% of the exam questions.