**Introduction to Contracts**

A **contract** is an agreement between two or more parties to do or not do something specific.

#### **Purpose of Contracts**

1. **Defines the Agreement**: Clearly outlines what each party agrees to.
2. **States the Goals**: Explains what both sides aim to achieve.
3. **Manages Issues**: Provides solutions for any problems that may arise during the contract.
4. **Termination Terms**: Explains how the contract can end and what happens after termination.

This helps both parties stay aligned and avoid misunderstandings.

If a contract is too harsh or unfair and causes issues between the parties, **contract law** ensures that such problems are resolved according to legal rules.

#### **What Contract Law Does**

1. **Termination Rules**: Provides guidelines for ending a contract if it becomes impossible to fulfill.
2. **Cancels Unfair Contracts**: Rejects contracts that are too harsh or unreasonable (unconscionable).

#### **Disputes in Contracts**

* Disputes in contracts happen when there is a disagreement about the terms or actions in the agreement. This can occur if one party doesn't fulfill their obligations or if there is confusion about what was agreed upon.
* Example: Imagine a ship carrying cargo. The contract ensures the cargo is delivered on time and in good condition. If the ship gets delayed or the cargo is damaged, the contract helps decide how to resolve the issue—like offering compensation or adjusting terms

#### **How to Avoid Disputes**

To prevent problems, it’s best to create a clear **contract document** that includes:

1. **Terms of Work**: Specifies the roles and responsibilities of both parties.
2. **Payment Methods**: Outlines how and when payments will be made.
3. **Termination Process**: Defines the proper way to end the contract, including any **notice period** required.

This helps both parties work efficiently and reduces the chances of future disagreements.

**Introduction to E-commerce and Contract Law**

E-commerce is growing quickly, and laws are evolving to keep up, like accepting digital signatures for online contracts. The internet has made the market global, so there’s a need for international laws to be more aligned. The EU is working on making trade easier by focusing on these areas:

1. **Electronic Signatures**: Using digital signatures to approve online contracts.
2. **Electronic Commerce**: Buying and selling goods/services online.
3. **Distance Contracts**: Agreements between people or businesses not in the same place.
4. **Distance Selling of Financial Services**: Selling financial products (like insurance or loans) online or over the phone.

### **Challenges in Tech-Related Contracts**

Many lawyers don’t know much about technology, and tech experts aren’t familiar with the law. This makes it hard to work together since both use complex words. Hilary Pearson said that optimists (positive thinkers) make good deal-makers, while pessimists (problem-finders) write better contracts by spotting future risks. When writing contracts, lawyers take a pessimistic approach to prevent future problems, which can annoy business clients who just want to close deals quickly. This mix of optimism and pessimism helps ensure deals are both exciting and safe.

* **Optimists** (positive thinkers) make good deal-makers.
* **Pessimists** (problem-finders) write better contracts by spotting future risks.

### **Introduction to Clear Contracts**

A **contract** should be easy to understand, consistent, and to the point. It must follow a **clear structure** and cover everything necessary without contradictions.

* **No ambiguity**: Both parties must know exactly what their **rights and responsibilities** are.
* **Avoid confusion**: If the contract is unclear, it can lead to **poor performance** and **disagreements**.
* **Disputes cost time and money**: Resolving issues caused by unclear contracts takes effort and resources.

### **Contracts Software Engineers Encounter**

Software engineers may deal with several types of contracts, such as:

1. **Insurance contracts** protect against financial losses.
2. **Employment contracts** set job terms and pay.
3. **Contracts with hardware suppliers** specify terms for buying equipment.
4. **Consultancy contracts** outline services provided by experts.

### **Contracts for Custom-Built Software**

Creating a good **contract** can be expensive because **commercial lawyers** charge high fees. To save time and money, **software suppliers** often use **standard form contracts**—templates used for multiple projects.

### **Structure of the Contract**

1. **Introductory Section**: Lists the names of the parties involved and basic details.
2. **Standard Terms and Conditions**: These remain the same for all projects.
3. **Appendices/Annexes**: Additional documents containing project-specific details.

**Appendices** and **Annexes** are both **attachments** to a contract, but:

* **Annexes**: Provide new, related information that adds to the contract.
* **Appendices**: Offer additional details (like data, charts, or examples) that support the main content of the contract

### **Key Issues in Standard Terms and Conditions**

1. **What is to be produced?**: Defines the software or service being developed.
2. **What is to be delivered?**: Lists the items the supplier must provide (software, documentation, etc.).
3. **Ownership of Rights**: Clarifies who owns the intellectual property.
4. **Payment Terms**: Explains when and how payments will be made.
5. **Delays and Changes**: Defines how to calculate payments if the project is delayed or changes occur.
6. **Penalty Clauses**: Outlines fines if deadlines or terms are not met.
7. **Client’s Obligations**: Specifies what the client must provide (e.g., access to data or resources).

### **Operational Issues Covered in Standard Terms**

1. **Standards and Methods**: Specifies the quality and working standards.
2. **Progress Meetings**: Defines how often and when meetings will occur.
3. **Project Managers**: Identifies who will oversee the project.
4. **Acceptance Procedure**: Explains how the client will approve the final product.
5. **Warranty and Maintenance**: Details any support or repairs after delivery.
6. **Termination**: Describes how and under what conditions the contract can end.

**What is to be Produced?**The contract explains what software or service will be made. A separate document called a requirements specification lists what the client wants. This specification should be clear and complete, but the client's needs may change during the project.

**Solution for Evolving Requirements:**Use flexible agreements to handle changes smoothly as the client’s needs evolve.

**What is to be Delivered?**Creating software is more than just giving the client a working program. The contract specifies what the client will receive, which may include:

* Source code: The raw program code.
* Command files: Scripts to build and install the program.
* Documentation: Information about the design and code.
* Manuals: Guides for users and operations.
* Software tools: Programs for maintenance.
* User training: Teaching users how to use the software.
* Maintenance training: Training for the client's IT staff.
* Test data and results: Proof that the software works.

**Ownership of Rights  
The contract must specify the legal rights the client gets regarding the software.**

* **Assignment**: The software house sells the copyright (ownership) to the client. The client gets full control over the software, but this usually happens only after the client makes the full payment.
* **License**: The software house keeps the copyright (ownership) but allows the client to use the software. The client can use it but doesn’t have full control over

**A license** is a legal agreement that gives someone permission to use someone else's property, typically intellectual property, under specific conditions. It does not transfer ownership but allows usage according to agreed terms

**Types of Licenses**

1. Exclusive License: The client is the only one who can use the software.
2. Non-exclusive License: The software house can let others use it too.

**Key Points for License Agreements**

* Duration: How long the license lasts and how it can be ended.
* Transferability: Whether the client can give the license to someone else.
* Scope: Limits on where and how the software can be used.
* Confidentiality: Making sure the client doesn’t share the software details with outsiders.

**Risk of Supplier Liquidation:** If a software supplier shuts down,**("Shutdown" means the software supplier goes out of business or stops supporting the software, which can happen due to bankruptcy, company closure, or acquisition)** the client may not get updates or support for their software. To prevent this, they can use an escrow agreement. This means a copy of the software code is held by a trusted third party, like a lawyer. If the supplier goes out of business, the client can access this code to keep their software running.

**Confidentiality in Contracts**The client might need to share private business information with the software company, and the software company may also want to keep certain details about the program confidential. Both parties agree, through clear contract terms, to protect each other’s information, enabling a secure partnership.

### **Payment Terms and Deadlines**

Payments in software contracts are generally due within **30 days** of the invoice date. If the client delays payment beyond this period, the company has the right to **terminate the contract** or apply a **2% surcharge** above the standard bank lending rate. However, such strict actions are rarely enforced, as they can damage the relationship between the client and supplier. Maintaining goodwill is essential for project success, so companies often resort to these measures only in extreme situations.

### **Payment Structure and Milestones**

A well-defined **payment schedule** ensures smooth cash flow throughout the project. Typically, **15% of the total contract value** is paid upfront when the contract is signed. As the project progresses, the client makes **stage payments** at key milestones, totaling **65%** of the value before delivery. Once the software is accepted, the supplier receives another **25%**. The last **10%** is kept until the end of the warranty period, making sure the supplier is responsible for any problems that arise after delivery. This step-by-step payment method connects payments to the project’s progress and reduces financial risks for both sides.

**Managing Delays and Extra Work**

Delays and changes in software development often lead to extra costs. The contract should explain how additional payments are calculated if the client is late with information or requests changes. Daily staff rates should be listed in an annex, and any adjustments must be agreed on in progress meetings. This helps manage changes smoothly and avoid disputes.

### **Penalty Clauses for Delays**

Some contracts include penalty clauses to stop delays. These reduce the supplier’s payment by a fixed amount for every week the project is late, up to a limit. Suppliers don’t like these clauses because they can increase project costs at the start. If the penalties reach the limit, suppliers might lose motivation to finish, as most of their payment is already gone. Even with penalties, delays hurt profits, so suppliers usually try to deliver on time unless there are big technical problems.

### **Client’s Responsibilities**

The success of a software project also depends on the client fulfilling their responsibilities. The client must provide **documentation** on relevant activities and ensure **access to key staff**. They also need to offer **machine facilities** for development and testing, along with **office space, communication tools, and support** if the supplier’s staff works on-site. If the client fails to meet these obligations, they may be liable for **delay-related costs**. Clear communication and timely support from the client are essential to avoid project delays and ensure smooth collaboration.

### **Warranty and Maintenance Period**

After the software is delivered, most contracts offer a **warranty period**—typically **90 days**—during which any bugs or issues reported are fixed free of charge. The **length of the warranty** is negotiable; a shorter warranty reduces costs, while a longer one increases them. Once the warranty period ends, the supplier may offer **maintenance services** to address future issues. The client can request these services as needed, ensuring long-term support and system reliability.

### **Contract Termination and Ownership**

In certain circumstances, the contract may need to be **terminated** before project completion. This could happen if the client is taken over by another company with an existing system or if the client’s needs change. If the contract ends prematurely, the supplier must be paid for all the work done so far, along with compensation for the time spent reallocating staff to other projects. The contract should also address the **ownership of incomplete work**, ensuring both parties understand their rights in case of termination.

**Arbitration for Dispute Resolution**

Disputes in software contracts can be costly if taken to court. To avoid this, many contracts include an **arbitration clause**, where both parties agree to let an independent arbitrator resolve any disagreements. Organizations like the **British Computer Society** or the **Institution of Electrical Engineers** provide qualified arbitrators with relevant technical expertise. However, some clients may hesitate to accept arbitration, fearing they might lose certain legal rights by doing so.

### **Inflation and Price Adjustments**

Long-term contracts include clauses to adjust prices for inflation. Suppliers can increase prices (e.g., yearly) to cover rising costs. The contract should say when and how these changes happen to protect both sides.

**Types of Software Service Contracts**:

1. **Contract Hire**: Supplier provides workers; the client pays daily rates. Workers follow the client’s directions.
2. **Time and Materials**: The client pays for time and materials used. Costs can increase if the project takes longer.
3. **Consultancy Contracts**: Supplier gives advice and recommendations, usually in a report. No software is made.

## **Strategies for Engineering Quality Software**

### **Importance of High-Quality Software**

* **Performance:** Software should run quickly and efficiently.
* **Safety:** It must operate reliably without failures.
* **User Needs:** Software should meet the needs of its users.

### **Fields Requiring Quality Software**

* Air Traffic Control, Nuclear Power, Automobile Safety, Health Care, Military and Defense etc

### **Demand for High-Quality Software**

* **User Expectations:** End users expect software to be stable, productive, and secure.
* **Consequences of Poor Quality:** Software defects can lead to system failures, loss of work, and security vulnerabilities.

### **Understanding Software Quality**

* **Software Defect:** A flaw in software that may stop it from meeting user needs, with impacts ranging from minor to major.
* **Software Quality:** The degree to which software fulfills user requirements.

### **Quality Management**

* **Goal:** To set, check, and improve the quality of how software is developed and the final product.
* **Deliverables:** Products include:
  + Statements of requirements
  + Flowcharts
  + User documentation

### **Causes of Poor Software Quality**

* **Lack of Knowledge:** Many developers do not know how to incorporate quality into software from the beginning or do not take the necessary time.
* Poor software quality often happens because developers may not know how to build quality into their work or are rushed due to tight deadlines. Sometimes, they skip thorough testing, which lets bugs slip through. Also, unclear requirements can cause the software to miss what users actually need. These problems lead to software that isn’t as reliable or useful as it should be
* **Developer Responsibilities:**
  + Follow strict engineering principles.
  + Learn from previous mistakes.
  + Understand the environment in which the system will operate.
  + Design systems that minimize human error.

### **Challenges in Software Development**

* **Programming Mistakes**: Developers commonly make errors, averaging one mistake for every 7-10 lines of code. These small errors can lead to larger issues if not found and fixed.
* **Time Pressure**: Developers often face intense deadlines to release software quickly, as companies want to start earning revenue as soon as possible. This rush can lead to skipping or reducing quality checks, which means defects are more likely to slip through.

### **Ethical Considerations**

* **Cost vs. Quality:** There's a tough decision about how much time and money to spend to make sure the software meets customer needs.
* **First Releases:** Companies often avoid buying the first version of new software or limit its use in important tasks because it might have issues.

### **Established Software Products**

* Even software that has been tested and works well can fail if operating conditions change unexpectedly.
* **Example:** Imagine a popular weather app that works perfectly under normal conditions. However, during a major storm, when many people are using it at once, the app might crash because it wasn't designed to handle so much traffic. This shows how even well-tested software can struggle if something unexpected happens.

## **The Importance of Software Quality**

* A **business information system** is a setup of different parts that work together to manage data for a company. It includes **hardware** (like computers), **software** (programs), **databases** (where data is stored), **networks** (connections between computers), **people** (users), and **procedures** (steps to use the system). The purpose of this system is to take raw data and turn it into useful information that helps the business make decisions.

### **Examples of Business Information Systems**

* **Order Processing System:** Used by manufacturers to handle customer orders.
* **Electronic Funds Transfer System:** Used by banks for transferring money electronically.
* **Online Ticket Reservation System:** Used by airlines to manage flight bookings.
* **Decision Support System (DSS):** Helps improve decision-making in organizations.
* **Industrial Control Software:** Manages industrial processes and controls many products we use.
* **Risks of Poor Software Quality** include significant negative impacts on a business. For example, if a banking app has bugs that prevent customers from accessing their accounts, it can lead to frustrated users and loss of trust. This not only disrupts daily operations but also damages the company's **reputation**, potentially resulting in long-term financial losses.

### **Ethical Questions**

* **Investment Decisions:** How much time and money should be spent to ensure software quality?
* **Product Liability:** What if the software causes harm? Companies face legal risks and must consider the potential damage.

## **Software Product Liability**

* **Software Product Liability** means that companies can be held responsible for problems caused by their software. If their software has bugs that lead to injuries or financial losses, they might get sued. For instance, if a software error in a medical device harms patients, the company could face legal action. This highlights the need for careful testing to ensure software works safely.

### **Notable Cases**

* **Samsung Galaxy Note 7 Recall**: Samsung had to recall the Galaxy Note 7 smartphones because the battery software caused the phones to overheat and catch fire. This led to a huge financial loss and damaged their reputation. It highlights how software issues can affect safety and lead to serious consequences.
* **Qantas Flight Incident:** In October 2008, a faulty onboard computer caused a Qantas flight to suddenly drop 8,000 feet, injuring 46 passengers.

**Legal Framework**: There isn’t a federal law for product liability; it mainly depends on state laws. If software has defects that cause harm, companies can be sued for:

* **Injury or death**: If a software error leads to someone getting hurt.
* **Loss of revenue**: If a business loses money because the software fails.
* **Increased costs**: If fixing the software causes unexpected expenses.

### **Basis for Software Liability Claims**

1. **Strict Liability**: A company must pay for any harm caused by bad software, even if they tried to be careful. People only need to show that the software was dangerous and that it caused the injury. Everyone involved in selling the software, like makers and sellers, can be held responsible.
2. **Negligence**: This is when a company doesn’t act carefully, like a responsible person would. They might make a mistake or forget to do something important. They’re only responsible for problems they should have noticed and fixed with normal care. They can defend themselves by showing they had a good reason for the mistake or that the other person also caused part of the problem.
3. **Breach of Warranty**: A **warranty** is a promise that a product will work well. If it doesn't, it's called a **breach**. The person claiming a problem has to show that the promise was broken. This can be difficult because warranties often say how much money the company has to pay for problems.
4. **Intentional Misrepresentation**: This happens when a seller lies about how good a product is or hides a problem. They can mislead people through ads, comments from salespeople, invoices, or shipping labels. Software companies often use limited warranties and disclaimers to protect themselves from these claims.

## **Software Development Process**

### **Roles in Large Software Projects**

* **System Analysts:** Gather requirements and analyze user needs.
* **Programmers:** Write and maintain the code.
* **Architects:** Design the overall structure of the software.
* **Database Specialists:** Manage data storage and retrieval.
* **Project Managers:** Oversee the project timeline, resources, and coordination.
* **Documentation Specialists:** Create user manuals and documentation.
* **Trainers:** Educate users on the software.
* **Testers:** Ensure the software works as intended through testing.

### **Software Development Methodology**

* Software Development Methodology is a set way to create software that helps keep everything organized. It outlines the steps to follow during development, assigns tasks to people or teams, suggests techniques for completing tasks, and gives rules to ensure the software is of good quality at every stage. This process makes sure that everything moves along smoothly and on schedule.

### **Importance of Early Problem Identification**

* Finding and fixing software issues early is much cheaper than fixing them later. It can cost up to 100 times more to fix a problem after the software is released. Detecting errors early saves money and improves software quality effectively.

### **Legal Protection and Quality Assurance**

* Following a good development methodology helps protect organizations from legal liability and reduces software errors.
* **Software Quality Assurance (QA):** Methods applied throughout the development cycle to ensure reliable operation.
  + QA methods are implemented at every stage and include testing before product release.

### **Types of Testing**

1. **Dynamic Testing:**
   * **Black-Box Testing:** The tester has no knowledge of the internal code structure.
   * **White-Box Testing:** The tester has complete knowledge of the internal logic and tests all possible paths in the code.
2. **Static Testing:**
   * Method of checking software for errors without executing the code, often through reviews and analysis of documents and code.
3. **Integration Testing:**
   * Performed after successful unit testing.
   * Combines software units into an integrated subsystem to ensure all linkages work correctly.
4. **System Testing:**
   * Conducted after integration testing.
   * Tests the entire system as a complete entity to ensure overall functionality.
5. **User Acceptance Testing:**
   * Independent testing done by trained end users to verify that the system meets their expectations.

## **Capability Maturity Model Integration (CMMI)**

* CMMI is a system created by the Software Engineering Institute at Carnegie Mellon University. It helps companies improve the way they work. The main goal is to guide businesses in making better software and managing projects more effectively. CMMI provides best practices to check current processes, find improvements, and boost quality, efficiency, and performance.

**Key Features of CMMI**:

1. **Five Levels of Maturity**: CMMI has five levels that show how advanced an organization is in software development. These levels help measure and improve current practices.
2. **Focus on Quality and Improvement**: CMMI emphasizes the importance of software quality. It highlights key issues that affect quality and provides steps to improve processes.

**Assessment and Improvement Steps in CMMI**:

1. **Conduct Assessment**: Organizations evaluate how they develop software to find out their current maturity level using the CMMI framework.
2. **Identify Areas for Improvement**: From the assessment, they pinpoint specific areas where they can make improvements.
3. **Define Action Plans**: They create plans to enhance their development processes and advance to a higher maturity level.

### **Benefits of Increasing Maturity**

* As organizations move up the maturity levels, they get better at delivering quality software on time and within budget. **CMMI-Development** has guidelines for **22 process areas** that are important for developing systems. When organizations follow these guidelines successfully, they can create and maintain software more effectively.

## **Key Issues in Software Development**

### **Consequences of Software Defects**

* **Deadly Outcomes:** Software defects in critical systems can lead to serious consequences, including injury or death.
* **Special Precautions:** Companies must implement additional precautions to ensure safety.

### **Ethical Decisions**

* **Trade-offs:** Ethical decisions often involve balancing quality with cost, ease of use, and time to market.

## **Development of Safety-Critical Systems**

### **Definition**

* **Safety-Critical Systems**: Systems where failure can cause serious harm, like nuclear reactors, airplane navigation, roller coasters, elevators, and medical devices.

### **Key Assumptions**

* **Rigorous Development Required:** Just using standard development methods isn't enough for safety-critical systems. These systems need more detailed and careful processes, which take more time to ensure they are safe and reliable

### **Development Process**

* **Additional Steps:** Development of safety-critical systems requires:
  + More thorough documentation
  + Rigorous checks and rechecks

### **Safety Management**

* **Project Safety Engineer:** This person makes sure the system is safe and follows safety rules.
* **Hazard Logging:** They keep track of potential dangers during the project by using a hazard log at every stage of development.

### **Safety Reviews**

* **Continuous Evaluation:** Regular safety reviews during the development process to ensure safety standards are met.
* **Configuration Management:** A robust system is necessary to track all safety-related documentation.

### **Ethical Dilemmas**

* Ethical challenges arise regarding:
  + Increased time and costs associated with safety processes.
  + Determining when Quality Assurance (QA) has completed sufficient testing.

## **Risk Management**

* **Risk:** The chance of something bad happening, based on how likely it is and how big the impact would be.
* **Consequences of Risk:** Can include damage to property, money loss, injury, or even death.

**Redundancy:**

* **Multiple Components:** Redundancy means having extra parts that do the same job to prevent problems if one part fails.
* **Historical Example:** In the 2013 blackout in the northeastern United States, some power grids failed due to a lack of backup systems, leading to widespread outages.

### **N-Version Programming**

* **Form of Redundancy:** Involves running different algorithms on two systems simultaneously to ensure reliability.
* **Error Correction:** Results from both systems are compared; if discrepancies arise, another algorithm is executed to determine the correct result.

### 

### **Risk Acceptance**

* **Decision-Making:** Organizations need to choose how much risk they can live with, and this can be a difficult choice.
* **System Modifications:** If the risks are too high, they may need to change the system to make it safer.
* **Emergency Procedures:** They also need to have plans ready for emergencies and recalls if something goes wrong.

## **Reliability and Human Interface**

* **Reliability Definition:** Reliability is the chance that a system or part will work correctly without breaking for its entire lifespan.
* **Human Interface Design:** Designing how humans interact with the system is very important for safety; if it's poorly designed, operators can make mistakes.
* **Case Study:** The USS Vincennes incident shows how bad interface design and human mistakes can lead to serious problems, highlighting the need for user-friendly and mistake-proof designs

## **Quality Management Standards**

ISO 9001 Family of Standards: These standards help companies ensure they provide quality products and services. Businesses must document and follow their processes, then prove to an outside reviewer they do it correctly.

Example: A bakery follows a written procedure for making bread, ensuring it’s always of good quality. They also keep records to show an auditor they follow the steps consistently.

### **Failure Mode and Effects Analysis (FMEA)**

* Failure Mode and Effects Analysis (FMEA): FMEA is a method to check how reliable a system is and identify what might go wrong if equipment fails. It ranks failures based on their impact on project success, safety, and customer satisfaction. The goal is to spot problems early so they can be fixed in time.
* Real-World Application: LifeScan, part of Johnson & Johnson, uses FMEA to ensure its OneTouch Systems are reliable for managing diabetes. They apply FMEA to test the software that automates blood glucose meter testing and assembly, ensuring the products are accurate and safe.