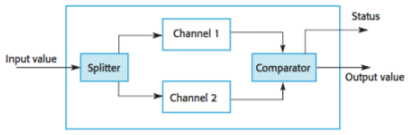
Safety - A property of a system. It is the system's ability to operate services (e.g. prevent danger causing human injury or death, and avoiding damage to the system's environment). Software safety issues become important. For example, most devices incorporate software-based control systems. Also, it can be used to control real-time, safety-critical processes.
Software in Safety-Critical Systems - 1) Software Controlled Systems - Decisions are made by the software. Subsequent actions are safety-critical. Software behavior is related to safety of the system. 2) Checking and monitoring safety-critical components (e.g. monitoring aircraft engine components for fault detection). 3) Monitoring software is safety-critical (e.g. other components may fail due to the failure of fault detection).
Safety & Reliability - Reliability and availability are not sufficient for system safety. Reliability - Conformance to a given specification and delivery of service. Safety - Ensuring system cannot cause damage. System reliability is essential for safety, however, reliable systems can be unsafe.
Unsafe Reliable Systems (4) - 1) Dormant System Faults - Undetected for number of years and only rarely arise. 2) Specification Errors - Software system behaves as specified but cause an accident. 3) Hardware Failures at Runtime - Generating spurious inputs. It is hard to anticipate in the specification. 4) Context-Sensitive Commands - For example, a system command is executed at the wrong time.
Safety-Critical Systems - It is essential that systems operation is always safe. They must not cause damage to people or the system's environment. For example: 1) Process control systems in chemical manufacturers. 2) Automobile control systems such as braking management systems.
Primary Safety Critical Systems -Cause associated hardware failures, directly threatening people (e.g. embedded software systems like an insulin pump control system). Insulin pump control systems collect data from a blood sugar sensor and calculates the amount of insulin required to be injected. Calculation is based on the rate of change of blood sugar levels. It sends signals to a micro-pump to deliver the correct dose of insulin. Safety-critical system as low blood sugars can lead to brain malfunctioning, coma, and death. High-blood sugar levels have long-term consequences such as eye and kidney damage.
Secondary Safety-Critical Systems - Result in faults in other connected systems, affecting safety consequences (e.g. mentcare system producing inappropriate treatment being prescribed). Infrastructure control systems.
Hazard - Situations or evens that can lead to an accident (e.g. incorrect computation by software in navigation system, or failure to detect possible disease in medication prescribing system). One should perform accident prevent actions as hazards do not inevitably lead to accidents.
Hazard Avoidance (3) - 1) Hazard Avoidance - Applying hazard avoidance design to software systems. Prevent come classes of hazards. 2) Hazard Detection and Removal - Detecting and removing hazards before causing accidents. 3) Damage Limitation - Protection features to minimize the damage.
Safety Terminology (6) - 1) Accident or Mishap - An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment. An overdose of insulin for example. 2) Hazard - A condition with the potential for causing or contributing to an accident. A failure of the sensor the measures blood glucose is an example. 3) Damage - A measure of the loss resulting from a mishap. Damage can range from many people being killed to minor injury or property damage. Damage resulting from an insulin overdose can be serious injury or death. 4) Hazard Severity - An assessment of the worst possible damage that could result from a particular hazard. When an individual death is a possibility, a reasonable assessment of hazard severity is "very high". 5) Hazard Probability - The probability of the events occurring which create a hazard. Probability values tend to be arbitrary but range from "probable" (1 out of 100) to "implausible" (no conceivable situations are likely to occur). 6) Risk - A measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the severity, and the probability that the hazard will lead to an accident.
Normal Accidents - Result have a single cause in complex systems which should be designed to be resilient to a single point of failure. A fundamental principle of safe systems design is that a single point of failure does not cause an accident. Thus, normal accidents are a result of a combination of malfunctions. It is hard to anticipate all combinations in software systems: 1) It is difficult to achieve complete safety. 2) Accidents are inevitable.
Software Safety Benefits (3) - Software control systems contribute to system safety: 1) A large number of conditions to be monitored and controlled. 2) Reducing human effort and time in hazardous environments. 3) Detecting and repairing safety-critical operator errors.
Safety Requirements (Functional & Non-Functional) - Functional - Statements of services the system should provide. For example, how the system should react to particular inputs and how the system should behave in particular situations. Non-Functional - Constraints on the services or functions of the system. Apply to the whole system rather than individual features.
Safety Functional Requirements - Describe functionality or systems services and depend on the type of software systems and users. Functional User Requirements - High-level statements of what the system should do. Functional System Requirements - The system services in detail.
Safety Specification - Goal is to identify protection requirements and prevent injury or death or environmental damage. Safety requirements are "Shall Not" requirements. They define situations and events that should never occur. Functional Safety Requirements - Checking and recovery features in a system. Protection features against failures and external attacks.
Hazard Driven Analysis (4) - 1) Hazard Identification, 2) Hazard Assessment, 3) Hazard Analysis, 4) Risk Reduction (by means of a safety requirements specification).
Hazard Identification - Identify the hazards threatening the system. Different types of hazards include physical, electrical, biological, and service failures).
Insulin Pump Risks (7) - 1) Insulin Overdose (Service). 2) Insulin Underdose (Service). 3) Power Failure due to Exhausted Battery (Electrical). 4) Electrical Interference with other Medical Equipment (Electrical). 5) Poor Sensor and Actuator Contact (Physical). 6) Infection Caused by Introduction of Machine (Biological). 7) Allergic Reaction to Materials or Insulin (Biological).
Hazard Assessment - Understanding the likelihood that a risk will arise and the potential consequences. Risk Categories: 1) Intolerable - Unsupportable. 2) As Low As Reasonably Possible (ALARP) - Minimizing risk possibility given available resources. 3) Acceptable - No extra costs to reduce hazard probability.
Social Acceptability of Risk - The acceptability of risk considering human, social and political considerations. Society is less willing to accept risk in most cases (e.g. the costs of cleaning up or preventing pollution). Subjective assessment depending on evaluators making the assessment.
Hazard Assessment (cont) - The risk probability ad the risk severity. Relative values: "unlikely", "rare", "very high", etc. It's impossible to do a precise measurement. Goal is to prevent or remove potential risks with the high severity.
Insulin Pump Risk Classification (9) - Key) Hazard - Probability - Severity - Estimated Risk - Acceptability. 1) Overdose Computation - Medium - High - High - Intolerable. 2) Underdose - Medium - Low - Low - Acceptable. 3) Failure of Hardware Monitoring - Medium - Medium - Low - ALARP. 4) Power Failure - High - Low - Low - Acceptable. 5) Machine Incorrectly Fitted - High - High - High - Intolerable. 6) Machine Breaks in Patient - Low - High - Medium - ALARP. 7) Machine Causes Infection - Medium - Medium - Medium - ALARP. 8) Electrical Interface - Low - High - Medium - ALARP. 9) Allergic Reaction - Low - Low - Low - Acceptable.
Hazard Analysis - The root causes of risks in a particular system. Hazard analysis techniques: 1) Inductive, Bottom-Up - Evaluate the hazards starting with system failures. 2) Deductive, Top-Down - Reason failure causes, starting with a hazard.
Fault-Tree Analysis - A deductive top-down technique. Hazard at the root of the tree and identify states causing hazards. Linking conditions by relationships (e.g. "and" and "or"). Goal is to minimize the number of single failure causes.
Fault-Tree Analysis (cont) - Possible conditions of incorrect dose of insulin: 1) Incorrect measurement of blood sugar level. 2) Failure of delivery system. 3) Dose delivered at wrong time. Root Causes of these hazards: 1) Algorithmic Error. 2) Arithmetic Error.
Risk Reduction - Goal is to identify requirements for risk managements to avoid accidents. Risk reduction strategies: 1) Hazard Avoidance. 2) Hazard Detection and Removal. 3) Damage Limitation.
Risk Reduction Strategy Use - Combining multiple risk reduction strategies (e.g. a chemical plant system. Detecting and correcting excess pressure in the reactor. Opening a relief valve as independent protection system).
Insulin Pump Software Risks (2) - 1) Arithmetic Error - Data variable overflow or underflow during a computation. Handling runtime exception. 2) Algorithmic Error – Comparison between previous and current values. Checking the maximum value to control dose.
Safety Requirements Examples (6) - 1) The system shall not deliver a single dose of insulin that is greater than a specified maximum dose for a system user. 2) The system shall not deliver a daily cumulative dose of insulin that is greater than a maximum daily dose for a system user. 3) The system shall include a hardware diagnostic facility that shall be executed at least 4 times per hour. 4) The system shall include an exception handler for all the exceptions that are identified. 5) The audible alarm shall be sounded when any hardware or software anomaly is discovered and a diagnostic message, as defined shall be displayed. 6) In the event of an alarm, insulin delivery shall be suspended until the user has reset the system and cleared the alarm.
Safety Engineering Processes - 1) Reviews and checks at each stage in the process. 2) General goal of fault avoidance and fault detection. 3) Safety reviews and explicit identification of hazards.
Regulation - Evidence that safety engineering processes are used. For example: 1) The specification and records of the checks. 2) Evidence of the verification and validation of the results. 3) Organizations for dependable software processes.
Agile Methods and Safety - Agile methods are not usually used for safety-critical systems engineering: 1) they require a lot of documentation. 2) a detailed safety analysis fo complete system specification is required. Despite this, test-driven development may be used.
Safety Assurance Processes - 1) Defining and ensuring a dependable process. 2) Process assurance focuses on whether processes are appropriate for dependability required and are followed by development team. 3) Should generate documentation about the safety of the system and thus agile processes are not sued for critical systems.
Processes for Safety Assurance - 1) Process assurance is important for safety-critical systems development as testing may not find all problems. Safety assurance activities include recording the analyses and personal responsibility.
Safety-Related Process Activities (5) - 1) A hazard logging and monitoring system. 2) Safety engineers who are responsible for safety. 3) Extensive use of safety reviews. 4) A safety certification system. 5) Detailed configuration management.
Hazard Analysis - Identifying hazards and their root causes. Traceability from identified hazards which is analysis to ensure hazards have been covered. A hazard log may be used to track hazards.
Safety Reviews - Are driven by the hazard register. They assess the system and judge whether to cope with hazards in a safe way.
Formal Verification - A mathematical specification of the system is produced. Static verification techniques are used in development: 1) A formal specification is mathematically analyzed for consistency and to discover specification errors and omissions. 2) Ensure a program conforms to its mathematical specification in to find programming and design errors.
Model Checking - Create a state model for a specialized system and check that model for errors. The model checker explores all possible paths and checks that a user-specified property is valid for each path. Verifying concurrent systems is helpful as they are hard to test. Model checking is computationally very expensive so verification of small to medium sized critical systems is only feasible.
Static Program Analysis - Uses tools for source text processing. Parses the program text to discover erroneous conditions. Effective as a supplement to inspections.
Automated Static Analysis Checks (5) - 1) Data Faults (e.g. variables used before initialization, declared but never used). 2) Control Faults (e.g. unreachable code, unconditional branches). 3) Input/Output Faults (e.g. variables output twice with no intervening assignment). 4) Interface Faults (e.g. parameter-type mismatches, parameter number mismatches). 5) Storage Management Faults (e.g. unassigned pointers, pointer arithmetic, memory leaks).
Levels of Static Analysis (3) - 1) Characteristic Error Checking - Check for patterns in the code for characteristic of errors. 2) User-Defined Error Checking - Define error patterns, ending error types by specific rules. 3) Assertion Checking - Formal assertion in their program. Symbolically executes to find potential problems.
Use of Static Analysis (3) - 1) Particularly valuable when a language has weak typing as many errors are undetectable by the compiler. 2) Security checking to discover areas of vulnerability such as buffer overflows. 3) The development of safety and security critical systems.
Safety & Dependability Cases - Safety and dependability cases are a structured set of documents. They provide evidence of a required level of safety or dependability. Regulators check a system is as safe or dependable. Regulators and developers work together for a system safety/ dependability case.
The System Safety Case - A safety case is a documented body of evidence that the system is adequately safe for a given environment. It is a formal proof, design rationale, and safety proof. Wider system safety case that takes hardware and operational issues into account.
The Contents of a Software Safety Case (10) - 1) System Description - An overview of the system and a description of its critical components. 2) Safety Requirements - Safety requirements abstracted from the system requirements specification. Details of other relevant system requirements may also be included. 3) Hazard and Risk Analysis - Documents describing the hazards and risks that have been identified and the measures taken to reduce risk. Hazard analysis and hazard logs. 4) Design Analysis - A set of structured arguments that justify why the design is safe. 5) Verification & Validation - A description of the V & V procedures used, and where appropriate, the test plans for the system. Summaries of the test results showing defects that have been detected and corrected. If formal methods have been used, a formal system specification and any analyses of that specification. Records of static analysis of the source code. 6) Review Reports - Records of all design and safety reviews. 7) Team Competences - Evidence of the competences of all of the team involved in safety-related systems development and validation. 8) Process QA - Records of the quality assurance processes carried out during system development. 9) Change Management Processes - Records of all changes proposed, actions taken, and where appropriate, justification of the safety of these changes. Information about configuration management procedures and configuration management logs. 10) Associated Safety Cases - References to other safety cases that may impact the safety case.
Structured Arguments - Safety cases should be based on structured arguments. Claims of safety and security should be justified by evidence.
Insulin Pump Safety Argument - 1) Claim: The maximum single dose of insulin to be delivered will not exceed a maximum dose. 2) Evidence: Safety argument for insulin pump. 3) Evidence: Test data for insulin pump. The value of current dose was correctly computed in 400 tests. 4) Evidence: Static analysis report for insulin pump software revealed no anomalies that affected the value of current dose. 5) Argument: The evidence presented demonstrates that the maximum does of insulin that can be computed is the maximum dose.
Structured Safety Arguments - Are structured arguments of a systems obligations. Generally based on a claim hierarchy.
Software Safety Arguments - Should show that the system cannot reach an unsafe state. These are weaker than correctness arguments which must show that the system code conforms to its specification. They are generally based on proof by contradiction: 1) Assume that the unsafe state can be reached. 2) Show that this is contradicted by the program code.
Insulin Safety Argument Program Paths - 1) Neither branch of it-statement 2 is executed. Can only happen if current dose is >= minimum dose and also <= maximum dose. 2) Then branch of if-statement 2 is executed. Current dose = 0. 3) Else branch of if-statement 2 is executed. Current dose = maximum dose. 4) In all cases, the post conditions contradict the unsafe condition that the dose administered is greater than the max dose.

Security Engineering - These are tools, techniques and methods to support the development and maintenance tasks and to resist malicious attacks to damage a system. It is a subfield of computer security.
Security Dimensions (3) - 1) Confidentiality - Information disclosed or made accessible. Data should not be shown to users who are not authorized to have access. 2) Integrity - Information should not be damaged which makes it unreliable. 3) Availability - It should be possible to access a system and its data. Can be complicated by denial of service attacks.
Security Levels (3) - 1) Infrastructure Security - The security of all systems and networks. A set of shared services to the organization. 2) Application Security - The security of individual or related groups of systems. 3) Operational Security - The secure operation of the organization's systems.
Application/Infrastructure Security - 1) Application Security - A system design problem in software engineering to resist attacks. 2) Infrastructure Security - Management problem to configure the system security infrastructure.

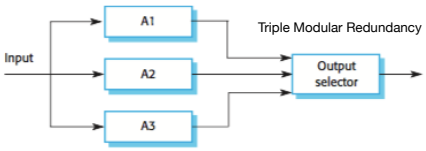
System Security Management (3) - 1) User and Permission Management - Managing system users based on appropriate permissions. 2) Software Deployment and Maintenance - Maintaining the application configuration to avoid vulnerabilities. 3) Attack Monitoring, Detection and Recovery - Monitoring unauthorized access, designing security strategies and developing backup and recovery strategies.
Operational Security - Prevent actions compromising system security such as insecure actions to make it easier to conduct operations. There is a trade-off between security and system effectiveness.
Security - The ability to protect from external attack. Networking across systems requires access to the system through the internet. Security is a prerequisite for availability, reliability and safety.
Fundamental Security - An insecure networked system. The system's reliability and safety are unreliable (e.g. an intrusion changes the running system or its data). This reduces the reliability and safety of the system.
Security Terminology (6) - 1) Asset - Something of value which has to be protected. The asset may be the software system itself or data used by that system. 2) Attack - An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage. 3) Control - A protective measure that reduces a system's vulnerability. Encryption is an example of a control that reduces a vulnerability of a weak access control system. 4) Exposure - Possible loss or harm to a computing system. This can be loss or damage to data, or loss of time and effort if recovery is necessary after a security breach. 5) Threat - Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack. 6) Vulnerability - A weakness in a computer-based system that may be exploited to cause loss or harm.
Examples of Security Terminology (6) - 1) Asset - The records of each patient that is receiving or has received treatment. 2) Exposure - Potential financial loss from future patients who do not seek treatment because they do not trust the clinic to maintain their data. Financial loss from legal action by the sports star. Loss of reputation. 3) Vulnerability - A weak password system which makes it easy for users to set guessable passwords. User ids that are the same as names. 4) Attack - An impersonation of an authorized user. 5) Threat - An unauthorized user will gain access to the system by guessing the credentials of an authorized user. 6) Control - A password checking system that disallows user passwords that are proper names that are normally included in a dictionary.
Threat Types (4) - 1) Interception Threats - Allows attacker to gain access to an asset (e.g. possible threat to gain access to patient records). 2) Interruption Threats - Make part of the system unavailable (e.g. a denial of service attack to make database connections impossible. 3) Modification Threats to Interfere with System Asset (e.g. Alter or destroy a patient record). 4) Fabrication Threats to Insert False Information (e.g. false transactions to transfer money to the perpetrator's bank account).
Security Assurance (3) - 1) Vulnerability Avoidance - The system designed without possibility of vulnerabilities (e.g. no network connection preventing external attacks). 2) Attack Detection and Elimination - The system designed to detect and remove vulnerabilities (e.g. virus checkers finding and removing viruses). 3) Exposure Limitation and Recovery - The system designed to minimize adverse results of an attack (e.g. a backup policy to restore damaged information).
Security and Dependability (4) - Security & Reliability - Reliability is affecting by security. Attacking systems or corrupting the system data hursts reliability. Resulting system failures compromise the reliability. Security & Availability - A common attack on a web-based system is a denial of service attack. These attacks service request which make the system unavailable.
Security is a Business Issue - Making security decisions in a cost-effective way (e.g. not spending more than the value of an asset). Should use a risk-based approach to support security decision. A security policy can be made based on security risk analysis. Security risk analysis is affected by practice aspects of business rather than a technical process.
Organizational Security Policies - Security Policies - General information access strategies for individual organizations. The point of security policies is to inform an organization about security and to be brief and precise technical documents. Security policy form a security engineering perspective should meet goals of the organization. Policies should be part fo the security engineering process.
Security Policies (4) - Should include: 1) The assets that must be protected. Security procedures should be applied to all organizations assets. All assets are not confidential. 2) The level of protection according to types of assets. A high level of security for sensitive personal information and a lower level for minor risks. 3) The responsibilities of individual users, managers and the organization. Should define expected user activities (e.g. strong passwords and office security). 4) Existing security procedures and technologies that should be maintained (e.g. existing security approaches despite known limitations due to a lack of maintenance costs).
Security Risk Assessment and Management - Risk assessment and management allows measuring possible losses resulted from system attacks and balancing these losses against the costs fo security procedures. There should be an organizational security policy. Risk management includes: 1) Preliminary Risk Management. 2) Life Cycle Risk Management. 3) Operational Risk Management.
Preliminary Risk Assessment - The goal is to identify generic risks to determine whether to achieve an adequate level of security at a reasonable cost. It is an analysis of high-level risks to the system. The outcomes of the risk assessment process are used to help identify security requirements.
Design Risk Assessment - The risk assessment is conducted during the system development life cycle. It produces the technical system design and implementation decisions. The results of the assessment are changes to the security requirements. Helps to identify known and potential vulnerabilities and specify functionalities and how to implement, test, and deploy software systems.
Operational Risk Assessment (3) - 1) Analyzing the operations of the system for possible risks resulted from human behavior. 2) Operational risk assessment evaluating continuously how the system is used by operators. 3) Organizational changes resulting in systems being used differently from original plans and new security requirements to be implemented.
Security Specification (4) - Shared with safety requirements specification to avoid incorrect results or unexpected behaviors in the systems. 4 Major Differences: 1) Safety problems in software operations are accidental (e.g. knowledge of security weaknesses exploited by attackers). 2) Identifying the root cause resulting in safety failures (e.g. the cause fo failure from a deliberate attack concealed by attackers. 3) A safety-related failure removed by the system-shutdown (e.g. system termination resulted fro the attack's intention). 4) System weaknesses examined and discovered by attackers (e.g. safety-related events resulted from incorrect software computation such as unplanned events leading to human death or injury or damage to property).
Types of Security Requirements (10) - 1) Identification. 2) Authentication. 3) Authorization. 4) Immunity. 5) Integrity. 6) Intrusion Detection. 7) Non-Repudiation. 8) Privacy. 9) Security Auditing. 10) System Maintenance Security.
Security Requirement Classification (3) - 1) Risk Avoidance Requirements - Design the system that risks cannot arise. 2) Risk Detection Requirements - Define mechanisms to identify the risk. 3) Risk Mitigation Requirements - Design the system to recover and restore system assets.
Security Risk Assessment (8) - 1) Asset Identification - Identify the key system assets to be protected. 2) Asset Value Assessment - Estimate the value of the identified assets. 3) Exposure Assessment - Assess the potential losses associated with each asset. 4) Threat Identification - Identify the most probable threats to the system assets. 5) Attack Assessment - Decompose threats into possible attacks. 6) Control Identification - Propose the controls to protect an asset. 7) Feasibility Assessment - Assess the technical feasibility and cost of the controls. 8) Security Requirements Definition - Security requirements for infrastructure or application system requirements.
Preliminary Risk Assessment Report for Mentcare System - 1) Asset - Information System. Value - High. Required to support all clinical consultations. Potentially safety-critical. Exposure - High. Financial loss as clinics may have to be cancelled. Costs of restoring the system. Possible patient harm if treatment not prescribed. 2) Asset - Patient Database. Value - High. Required to support all clinical consultations. Potentially safety-critical. Exposure - High. Financial loss as clinics may have to be canceled. Costs of restoring system. Possible patient harm if treatment cannot be prescribed. 3) Asset - Individual Patient Record. Value - Normally low although may be high for specific high-profile patients. Exposure - Low direct losses but possible loss of reputation.
Threat and Control Analysis in Preliminary Risk Assessment Report - 1) Threat - An unauthorized user gains access as system manager and makes system unavailable. Probability - Low. Control - Only allow system management from specific specific locations that are physically secure. Feasibility - Low cost of implementation but care must be taken with key distribution and to ensure that the keys are available in the event of an emergency. 2) Threat - An unauthorized user gains access as a system user and accesses confidential information. Probability - High. Control - Require all users to authenticate themselves using a biometric mechanism.Log all changes to patient information to tracking system usage. Feasibility - Technically feasible but high-cost solution. Possible user resistance. Simple and transparent to implement and also supports recovery.
Security Requirements in Mentcare System - 1) Patient information shall be downloaded at the start of a clinic session to a secure area on the system client that is used by clinical staff. 2) All patient information on the system client shall be encrypted. 3) Patient information shall be uploaded to the database after a clinic session has finished and deleted from the client computer. 4) A log on a separate computer from the database server must be maintained of all changes made to the system database.
Misuse Cases (4) - Misuse cases are instances of threats to a system. 1) Interception Threats - Attacker gains access to an asset. 2) Interruption Threats - Attacker makes part of a system unavailable. 3) Modification Threats - A system asset if tampered with. 4) Fabrication Threats - False information is added to a system.
Mentcare Misuse Case - Transfer Data - Actors - Medical receptionist, Patient record system (PRS). Description - A receptionist may transfer data form the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone, etc.) or a summary of the patient's diagnosis and treatment. Data - Patient's personal information and treatment summary. Stimulus - User command issued by medical receptionist. Response - Confirmation that PRS has been updated. Comments - The receptionist must have appropriate security permissions to access the patient information and the PRS.

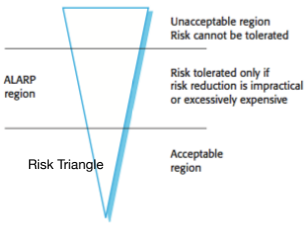
Mentcare Misuse Case - Intercept Transfer - Actors - Medical receptionist, Patient records system (PRS), Attacker. Description - A receptionist transfers data from his or her PC to the Mentcare system on the server. An attacker intercepts the data transfer and takes a copy of that data. Data/Assets - Patient's personal information and treatment summary. Attacks - A network monitor is added to the system and packets from the receptionist to the server are intercepted. A spoof server is setup between the receptionist and the database server so that receptionist believes they are interacting with the real system. Mitigations - All networking equipment must be maintained in a locked room. Engineers accessing the equipment must be accredited. All data transfers between the client and server must be encrypted. Certificate-based client-server communication must be used. Requirements - All communications between the client and server must use Secure Socket Layer (SSL). The https protocol uses certificate-based authentication and encryption.
Secure Systems Design - 1) Security should be designed into a system. It is difficult to make an insecure system secure after implementation. 2) Architectural design decisions affect the security of a system. 3) Must discuss what is considered good practice to design secure systems.
Design Compromises - Adding security features can affect other attributes. 1) Performance - Additional security checks slow down a system. 2) Usability - Security measures may require addition user interactions. Less usable and can frustrate system users.
Design Risk Assessment - Risk assessment can be performed during development and after it has been deployed. At this time, more information is available (e.g. system platform, middleware, architecture). Vulnerabilities may be identified during design processes.
Protection Requirements - Knowledge of information representations and systems distribution is available. May make decisions to separate patient and treatment information depending on amount of information to be protected. May also maintain copies of records on a local client to protect against denial of service attacks, but they should be encrypted.
Design Decisions form Use of COTS (3) - 1) Authenticated using a name/password combination. 2) Accessing the system through a standard web browser. 3) Information is presented as an editable web form.
Security Requirements (3) - 1) A password checker shall be made available and shall be run daily. Weak passwords shall be reported to system administrators. 2) Access to the system shall only be allowed by approved client computers. 3) All client computers shall have a single, approved web browser installed by system administrators.
Architectural Design - There are issues for an architecture design for security: 1) Protection - How to organize a system to protect critical assets. 2) Distribution - How to distribute system assets to minimize the effects of attacks. These both are potentially conflicting. If assets are distributed, then they are more expensive to protect. If assets are protected, performance may be compromised.
Protection Levels (3) - 1) Platform-Level - Top-level controls on the platform on which a system runs. 2) Application-Level - Specific protection mechanisms built into the application. 3) Record-Level - Protection is invoked when access to specific information. 4) These lead to a layered protection architecture.
Distribution - Important so that attacks do not lead to complete loss of system service. Each platform has separate protection features that are different from other platforms and do not share a common vulnerability. This mitigates the risk of denial of service attacks.
Design Guidelines for Security Engineering - Can encapsulate good practice in secure systems design. The purposes of design guidelines is to raise awareness of security issues. It is the basis of a review checklist during the validation process. Design guidelines are applicable during software specification and design.
Design Guidelines for Secure Systems Engineering - 1) Base security decisions on an explicit security policy. 2) Avoid a single point of failure. 3) Fail securely. 4) Balance security and usability. 5) Log user actions. 6) Use redundancy and diversity to reduce risk. 7) Specify the format of all system inputs. 8) Compartmentalize your assets. 9) Design for deployment. 10) Design for recoverability.
Design Guidelines (10) - 1) Base decisions on an explicit security policy (e.g. a policy for the organization and the security requirements). 2) Avoid a single point of failure (e.g. security failure can occur only if there is more than one failure). 3) Fail securely (e.g. sensitive data cannot be accessed by unauthorized users). 4) Balance security and usability (e.g. avoid security procedures that make the system difficult to use. 5) Log user actions (e.g. maintain a log of user actions to discover who did what. 6) Use redundancy and diversity to reduce risk (e.g. keep multiple copies of data and use diverse infrastructure to avoid the single point of failure). 7) Specify the format of all system inputs (e.g. check all inputs are within range to avoid unexpected inputs. 8) Compartmentalize your assets (e.g. organize the system assets in separate areas). 9) Design for deployment (e.g. design the system to avoid deployment problems). 10) Design for recoverability (e.g. design the system for recoverability after an attack).
Secure Systems Programming - 1) Design security into an application system. 2) Consider security during programming. 3) No bound checking overwrites the program stack.
Aspects of Secure System Programming - 1) Vulnerabilities are often language-specific (e.g array bound checking is automatic in most modern languages). Avoiding the use of languages is not a realistic option. 2) Security vulnerabilities are related to program reliability (e.g. programs without array bounds checking can crash). Improving reliability can also improve system security.
Security Testing - Testing the system can protect itself from attacks. Problems with security testing: 1) Impossible to define security requirements as simple constraints. 2) Attackers are intelligent and look for vulnerabilities. 3) Discover weaknesses and loopholes in the system.
Security Validation (4) - 1) Experience-based Testing - Review and analyze against the known types of attacks. 2) Penetration Testing - Simulating attacks on the system. 3) Tool-based Analysis - Various security tools are used to analyze the system operation. 4) Formal Verification - The system is verified against a formal security specification.
Examples of Entries in Security Checklist - 1) Do all files that are created in the application have appropriate access permissions? The wrong access permissions may lead to these files being accessed by unauthorized users. 2) Does the system automatically terminate user sessions after a period of inactivity? Sessions that are left active may allow unauthorized access through an unattended computer. 3) If the system is written in a programming language without array bound checking, are there situations where buffer overflow may be exploited? Buffer overflow may allow attacks to send code strings to the system and then execute them. 4) If passwords are set, does the system check that passwords are "strong"? Strong passwords consist of letters, numbers, punctuation, and are not normal dictionary entries. They are more difficult to break than simple passwords. 5) Are inputs from the system's environment always checked against an input specification? Incorrect processing of badly formed inputs is a common cause of security vulnerabilities.

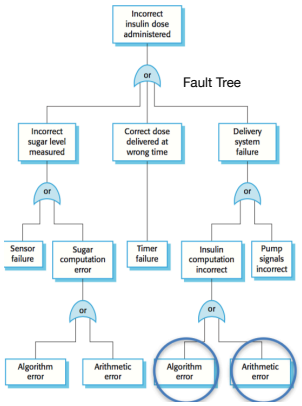
Self-Monitoring Arch.



Triple Modular Redundancy



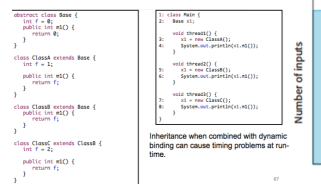


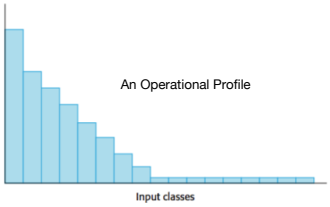


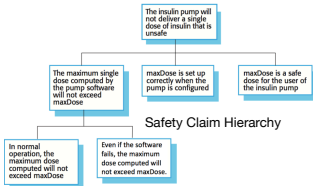
Fault Tree

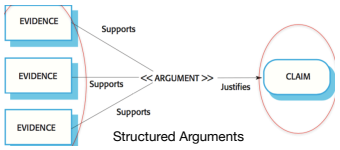
```
public static void main(String[] args) throws Exception {
  ClassPool pool = ClassPool.getDefault();

  // Insert Class Path
  pool.insertClassPath("/path/to/classpath"); // with string
  pool.insertClassPath(new ClassClassPath(Rectangle.class)); // with class

  // Set Superclass
  CtClass rectClass = pool.get("target.Rectangle");
  rectClass.setSuperclass(pool.get("target.Point"));
  rectClass.writeFile(); // can pass path c.writeFile("path/to/output");

  // New Field
  CtClass pointClass = pool.get("target.Point");
  CtField newField = new CtField(CtClass.intType, "z", pointClass);
  pointClass.addField(newField);

  // New Method
  CtMethod newMethod = CtNewMethod.make("void foo() { ... }", pointClass);
  pointClass.addMethod(newMethod);

  // New Class & Defrost
  CtClass newClass = pool.makeClass("NewRectangle");
  newClass.writeFile(); // can pass path c.writeFile("path/to/output");
  newClass.defrost(); // modifications now permitted
  newClass.setSuperclass(pointClass);
  newClass.writeFile();

  // toClass
  CtClass helloClass = pool.get("Hello");
  CtMethod sayMethod = helloClass.getDeclaredMethod("say");
  sayMethod.insertBefore("{ System.out.println(\"Hello.say\"); }");
  Class<Hello> c = helloClass.toClass();
  Hello h = c.newInstance();
  h.say();

  // Using Loader & Invoke
  Loader loader = new Loader(pool);
  Class<?> hClass = loader.loadClass("Hello");
  Object helloInstance = hello.newInstance();
  Class<?> hClass2 = helloInstance.getClass();
  Method m = hClass2.getDeclaredMethod("getVal", new Class[] {});
  m.invoke(helloClass, new Object[] {});

  // Retrieving Class Objects
  Rectangle rectangle = new Rectangle();
  rectangle.getClass().getField("hiddenValue").getName();
  Rectangle.class.getField("hiddenValue").getName();
}
```

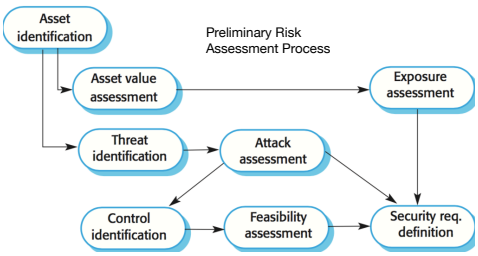
```
public class Main extends ClassLoader {

  public static void main(String[] args) throws Exception {
    Main sampleLoader = new Main();
    Class<?> c = sampleLoader.loadClass("MyApp");
    c.getDeclaredMethod("main", new Class[] { String[].class })
        .invoke(null, new Object[] { args });
  }

  private ClassPool pool;

  public Main() throws NotFoundException {
    pool = new ClassPool();
    pool.insertClassPath("/path/to/classpath");
  }

  @Override // Add Field
  protected Class<?> findClass(String name) throws ClassNotFoundException {
    try {
      CtClass c = pool.get(name);
      if (name.equals("MyApp")) {
        // New Field
        CtField newField = new CtField(CtClass.intType, "hiddenValue", c);
        newField.setModifiers(Modifier.PUBLIC);
        c.addField(newField);

        // $0 (this), $1 $2 (params), Control Flow - Depth of Recursive Calls
        CtMethod factMethod = c.getDeclaredMethod("fact");
        factMethod.useCflow("fact");
        factMethod.insertBefore("if ($cflow(fact) == 0) System.out.println(\"$0\");");
      }
      return defineClass(name, c.toBytecode(), 0, c.toBytecode().length);
    } catch (Exception e) {
      throw new ClassNotFoundException();
    }
  }
}
```

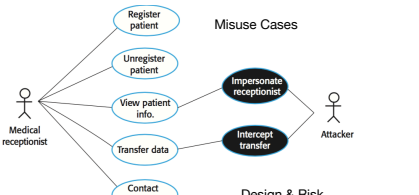


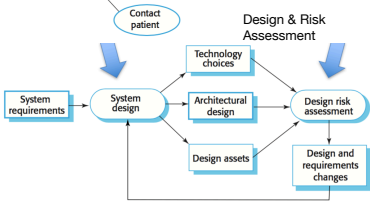


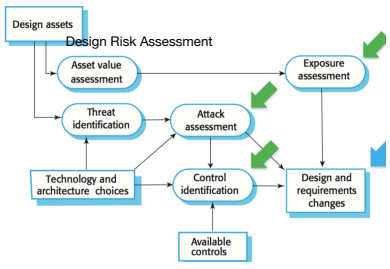


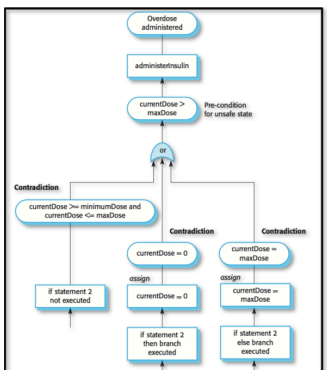






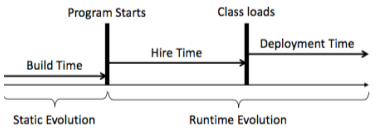


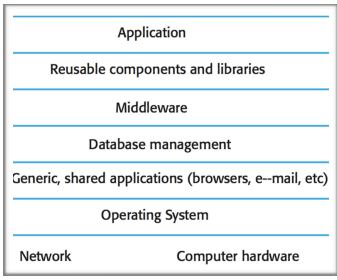




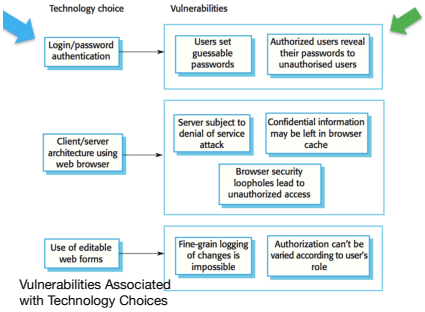
Formal

```
...

/* if statement 2 */
if ( currentDose < minimumDose ) {
    currentDose = 0 ;
}
else if ( currentDose > maxDose ) {
    currentDose = maxDose ;
}

administerInsulin (currentDose) ;
...
```

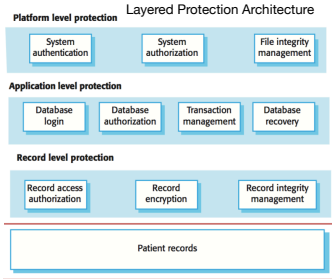




System Layers where Security can be Compromised



Vulnerabilities Associated with Technology Choices



Layered Protection Architecture