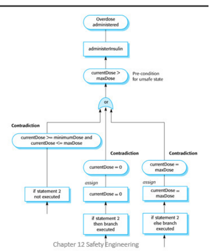
```java
public static void main(String[] args) {
        ClassPool pool = ClassPool.getDefault();
        boolean useRuntimeClass = true;
        if (useRuntimeClass) {
            ClassClassPath classPath = new
ClassClassPath(new Rectangle().getClass());
            pool.insertClassPath(classPath);
        } else {
            String strClassPath = workDir +
"\\bin";
            pool.insertClassPath(strClassPath);
        }
        CtClass cc = pool.get("target.Rectangle");

cc.setSuperclass(pool.get("Target.Point"));//takes
a CtClass
        cc.writeFile(outputDir);//outputDir is a
string

         ClassPool pool = ClassPool.getDefault();
         boolean useRuntimeClass = true;
         if (useRuntimeClass) {
           ClassClassPath classPath = new
ClassClassPath(new Rectangle().getClass());
            pool.insertClassPath(classPath);
         } else {
              String strClassPath = workDir +
         "\\bin";
               pool.insertClassPath(strClassPath);
         CtClass cc = pool.get("target.Rectangle");

curClass.setSuperclass(pool.get(superClass));
         cc.writeFile(outputDir);

         ClassPool pool = ClassPool.getDefault();
         CtClass cc = pool.makeClass(newClassName);
         cc.writeFile(outputDir);
         CtClass ccInterface =
pool.makeInterface(newInterfaceName);
         ccInterface.writeFile(outputDir);

         ClassPool pool = ClassPool.getDefault();
      String strClassPath = outputDir;
      pool.insertClassPath(strClassPath);
         CtClass ccPoint2 =
pool.makeClass("Point2");
         ccPoint2.writeFile(outputDir);
         CtClass ccRectangle2 =
pool.makeClass("Rectangle2");
         ccRectangle2.writeFile(outputDir);
         // ccRectangle2.defrost(); //
modifications of the class definition will be
permitted.
         ccRectangle2.setSuperclass(ccPoint2);
         ccRectangle2.writeFile(outputDir);

        CtMethod m = cc.getDeclaredMethod("say");
        m.insertBefore("{
System.out.println(\"Hello.say:\"); }");
        Class<?> c = cc.toClass();
        Hello h = (Hello) c.newInstance();
        h.say();

   private static String       workDir          =
System.getProperty("user.dir");
   private static final String TARGET_POINT     =
"target.Point";
   private static final String TARGET_RECTANGLE =
"target.Rectangle";
         ClassPool cp = ClassPool.getDefault();
      String strClassPath = workDir +
File.separator + "classfiles";
      pool.insertClassPath(strClassPath);
         Loader cl = new Loader(cp);
         CtClass cc = cp.get(TARGET_RECTANGLE);
         cc.setSuperclass(cp.get(TARGET_POINT));
         Class<?> c =
cl.loadClass(TARGET_RECTANGLE);
         Object rect = c.newInstance();
         System.out.println("[DBG] rect object: " +
rect);
```

```java
         Class<?> rectClass = rect.getClass();
         Method m =
rectClass.getDeclaredMethod("getVal", new Class[]
{});
         System.out.println("[DBG] method: " + m);
         System.out.println("[DBG] result: " +
m.invoke(rect, new Object[] {}));

   public static void main(String[] args) throws
Throwable {
      SampleLoader s = new SampleLoader();
      Class<?> c = s.loadClass("MyApp");
      c.getDeclaredMethod("main", new Class[] {
String[].class }).invoke(null, new Object[] { args
});
   }
   private ClassPool pool;
   public SampleLoader() throws NotFoundException {
      pool = new ClassPool();
      pool.insertClassPath(inputDir); //
MyApp.class must be there.
   }
public static void main(String[] args) throws
Throwable {
      SubstituteMethodBody s = new
SubstituteMethodBody();
      Class<?> c = s.loadClass(TARGET_MY_APP);
      Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
      mainMethod.invoke(null, new Object[] { args
});
   }

   protected Class<?> findClass(String name) throws
ClassNotFoundException {
      CtClass cc = null;
      try {
         cc = pool.get(name);
         cc.instrument(new ExprEditor() {
            public void edit(MethodCall m) throws
CannotCompileException {
               String className = m.getClassName();
               String methodName =
m.getMethodName();

               if (className.equals(TARGET_MY_APP)
&& methodName.equals(DRAW_METHOD)) {
                  System.out.println("[Edited by
ClassLoader] method name: " + methodName + ", line:
" + m.getLineNumber());
                  m.replace("{"//
                        + "$proceed($$); "//
                        + "}");
               } else if
(className.equals(TARGET_MY_APP) &&
methodName.equals(MOVE_METHOD)) {
                  System.out.println("[Edited by
ClassLoader] method name: " + methodName + ", line:
" + m.getLineNumber());
                  m.replace("{" //
                        + "$1 = 0; " //
                        + "$proceed($$); " //
                        + "}");
               }
            }
         });
         byte[] b = cc.toBytecode();
         return defineClass(name, b, 0, b.length);


static String workDir   =
System.getProperty("user.dir");
ClassPool pool = ClassPool.getDefault();
         pool.insertClassPath(inputDir);
         CtClass cc = pool.get("target.Point");
         CtMethod m = cc.getDeclaredMethod("move");
         m.insertBefore("{
System.out.println(\"[DBG] param1: \" + $1); " + //
               "System.out.println(\"[DBG] param2:
\" + $2); }");
```

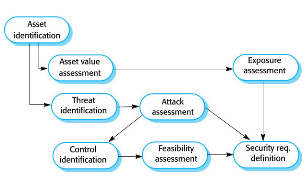
```java
         cc.writeFile(outputDir);
         System.out.println("[DBG] write output to:
" + outputDir);

        ClassPool defaultPool =
ClassPool.getDefault();
        defaultPool.insertClassPath(INPUT_PATH);
        CtClass cc = defaultPool.get(TARGET_MYAPP);
        CtMethod m =
cc.getDeclaredMethod(FACT_METHOD);
        m.useCflow(FACT_METHOD);
        m.insertBefore("if ($cflow(fact) == 0)" +
System.lineSeparator() + //
                "System.out.println(\"[MyAppFact
Inserted] fact \" + $1);");
        cc.writeFile(OUTPUT_PATH);
        InsertMethodBodyCflow s = new
InsertMethodBodyCflow();//pool = new
ClassPool();pool.insertClassPath(OUTPUT_PATH); //
TARGET must be there.
        Class<?> c = s.loadClass(TARGET_MYAPP);
        Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
        mainMethod.invoke(null, new Object[] { args
});
        //findClass method:cc =
pool.get(name);byte[] b = cc.toBytecode();return
defineClass(name, b, 0, b.length);

        SubstituteMethodBody s = new
SubstituteMethodBody();// pool = new
ClassPool();pool.insertClassPath(new
ClassClassPath(new
java.lang.Object().getClass()));pool.insertClassPat
h(INPUT_PATH); // "target" must be there.
        Class<?> c = s.loadClass(TARGET_MY_APP);
        Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
        mainMethod.invoke(null, new Object[] { args
});
        cc = pool.get(name);
        cc.instrument(new ExprEditor() {
           public void edit(MethodCall m) throws
CannotCompileException {
           }
         byte[] b = cc.toBytecode();
         return defineClass(name, b, 0, b.length);

           FieldAcess s = new FieldAcess();//pool
= new ClassPool();pool.insertClassPath(new
ClassClassPath(new
java.lang.Object().getClass()));pool.insertClassPat
h(INPUT_PATH); // TARGET must be there.
           Class<?> c =
s.loadClass(TARGET_MY_APP);
           Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
           mainMethod.invoke(null, new Object[] {
args });

           NewExprAccess s = new NewExprAccess();
           Class<?> c =
s.loadClass(TARGET_MY_APP2);
           Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
      mainMethod.invoke(null, new Object[] { args
});
           cc = pool.get(name);
         cc.instrument(new ExprEditor() {
              public void edit(NewExpr newExpr)
throws CannotCompileException {
              StringBuilder code = new
StringBuilder();
              code.append("\"y: \" + " +
"$_.y);\n }\n");
              // System.out.println(code);
              newExpr.replace(code.toString());
```

```java
         String src = "public void xmove(int dx) { x
+= dx; }";
         CtMethod newMethod = CtNewMethod.make(src,
cc);
         cc.addMethod(newMethod);
         cc.writeFile(outputDir);

         CtMethod newMethod = CtNewMethod.make(src,
cc, "this", "move");
         cc.addMethod(newMethod);
         cc.writeFile(outputDir);

         ClassPool pool = ClassPool.getDefault();
         pool.insertClassPath(inputDir);
         CtMethod newMethod = new
CtMethod(CtClass.intType, "move", new CtClass[] {
CtClass.intType }, cc);
         cc.addMethod(newMethod);
         newMethod.setBody("{ x += $1; return x;}");
         cc.setModifiers(cc.getModifiers() &
~Modifier.ABSTRACT); cc.writeFile(outputDir);

ClassPool pool = ClassPool.getDefault();
          pool.insertClassPath(inputDir);
          CtClass cc = pool.get("target.Point");
          String src = "public void xmove(int dx) {
x += dx; }";
          CtMethod newMethod = CtNewMethod.make(src,
cc);
          cc.addMethod(newMethod);
          cc.writeFile(outputDir);

          ClassPool pool = ClassPool.getDefault();
          pool.insertClassPath(inputDir);
          CtClass cc = pool.get("target.Point");
          String src = "public void ymove(int dy) {
$proceed(0, dy); }";
          CtMethod newMethod = CtNewMethod.make(src,
cc, "this", "move");
          cc.addMethod(newMethod);
          cc.writeFile(outputDir);
          System.out.println("[DBG] write output to:
" + outputDir);

          ClassPool pool = ClassPool.getDefault();
          pool.insertClassPath(inputDir);
          CtClass cc = pool.get("target.Point");
          CtMethod newMethod = new
CtMethod(CtClass.intType, "move", new CtClass[] {
CtClass.intType }, cc);
          cc.addMethod(newMethod);
          newMethod.setBody("{ x += $1; return
x;}");
          cc.setModifiers(cc.getModifiers() &
~Modifier.ABSTRACT);
          cc.writeFile(outputDir);

          ClassPool pool = ClassPool.getDefault();
          pool.insertClassPath(inputDir);
          CtClass cc = pool.get("target.Point");
          CtMethod m = CtNewMethod.make("public
abstract int m(int i);", cc);
          CtMethod n = CtNewMethod.make("public
abstract int n(int i);", cc);
          cc.addMethod(m);
          cc.addMethod(n);
          m.setBody("{ return ($1 <= 0) ? 1 : (n($1
- 1) * $1); }");
          n.setBody("{ return m($1); }");
          cc.setModifiers(cc.getModifiers() &
~Modifier.ABSTRACT);

CtField f = new CtField(CtClass.intType, "z",
pointClass);
          pointClass.addField(f);
```

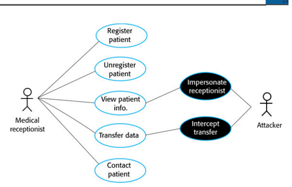
Informal safety argument based on demonstrating contradictions



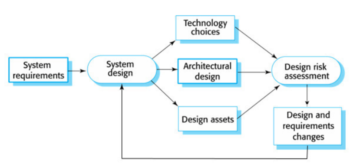
The preliminary risk assessment process for security requirements



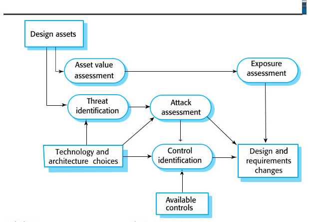
Misuse cases



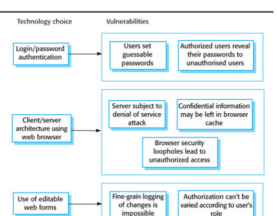
Design and risk assessment



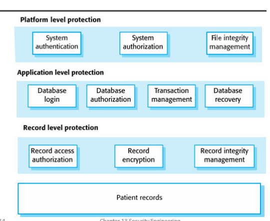
Design risk assessment

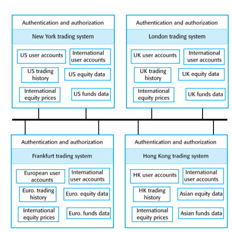


Vulnerabilities associated with technology choices



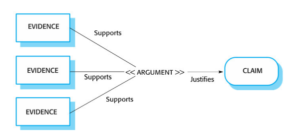
A layered protection architecture

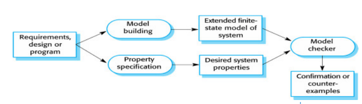




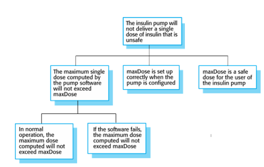
Distributed assets in an equity trading system

Structured arguments

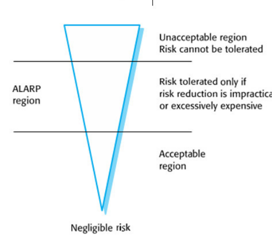


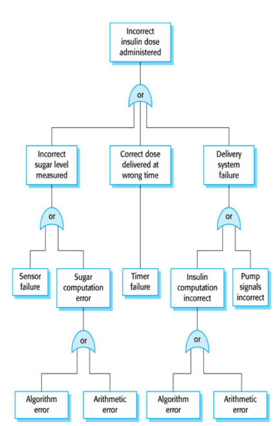


A safety claim hierarchy for the insulin pump



The risk triangle





A layered protection architecture — 1. Platform level protection a. system authentication b. system authorization, c. file integrity management 2. application level protection a. database login b. database authorization c. transaction management d. database recovery 3. record level protection a. record access authorization b. record encryption c. record integrity management 4. patient records

A safety claim hierarchy for the insulin pump -- insulin pump will not deliver single dose that is unsafe ●> (1. the max dose cmputed by software will not exceed x, 2. maxdose is set up correctly when the pump is configred 3. maxdose is a safe dose for the use of the insulin pump) ●> (1. in normal operation the max dose will not exceed maxdose 2. if the s/w fails the max dose will not exceed the max dose)

A simplified hazard log entry — System: Insulin Pump System Safety Engineer: James Brown Identified Hazard Insulin overdose delivered to patient Identified by Jane Williams Criticality class Identified risk High Fault tree identified Fault tree creators Jane Willliams and Bill Smith Fault tree checked Brown

Agile methods and safety — ↑Agile methods are not usually used for safety•critical systems engineering ↑ Extensive process and product documentation is needed for system regulation. Contradicts the focus in agile methods on the software itself. ↑ A detailed safety analysis of a complete system specification is important. Contradicts the interleaved development of a system specification and program. ↑Some agile techniques such as test•driven development

Application slash infrastructure security -- ↑Application security is a software engineering problem where the system is designed to resist attacks. ↑Infrastructure security is a systems management problem where the infrastructure is configured to resist attacks. ↑The focus of this chapter is application security rather than infrastructure security.

Architectural design -- ↑Two fundamental issues have to be considered when designing an architecture for security. ↑ Protection • How should the system be organisedso that critical assets can be protected against external attack? ↑ Distribution • How should system assets be distributed so that the effects of a successful attack are minimized? ↑These are potentially conflicting ↑ If assets are distributed, then they are more expensive to protect. If assets are protected, then usability and performance

Arguments against formal methods — ↑Require specialized notations that cannot be understood by domain experts. ↑Very expensive to develop a specification and even more expensive to show that a program meets that specification. in a program more cheaply using other V & V techniques.

Arguments for formal methods — ↑Producing a mathematical specification requires a detailed analysis of the requirements and this is likely to uncover errors. ↑Concurrent systems can be analysed to discover race conditions that might lead to deadlock. Testing for such problems is very difficult. ↑They can detect implementation errors before testing when the program is analyzed alongside the specification.

Aspects of secure systems programming -- ↑Vulnerabilities are often language•specific. ↑ Array bound checking is automatic in languages like Java so this is not a vulnerability that can be exploited in Java programs. ↑ However, millions of programs are written in C and C++ as these allow for the development of more efficient software so simply avoiding the use of these languages is not a realistic option. ↑Security vulnerabilities are closely related to program reliability. ↑ Programs without array bound checking can crash so actions taken to improve program reliability can also improve system security.

Asset analysis in a preliminary risk assessment — report for the Mentcare system Asset Value Exposure TheinformationsystemHigh. Required to support all clinical consultations. Potentially safety•critical. High. Financial loss as clinics of restoring system. Possible patient harm if treatment cannot beprescribed. ThepatientdatabaseHigh. Required to support all clinical consultations. Potentially safety•critical. High. Financial loss as clinics of restoring system. Possible patient harm if treatment cannot beprescribed. high for specific high•profile patients. Low direct losses but possible lossofreputation.

Automated static analysis checks — Fault class Static analysis check Data faults Variables used before initialization Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables Control faults Unreachable code Unconditional branches into loops Input slash output faults Variables output twice with no intervening assignment Interface faults Parameter•type mismatches Parameter number mismatches Non•usage of the results of functions Uncalled functions and procedures Storage management faults Unassigned pointers Pointer arithmetic Memory leaks

Balance security and usability — ↑ Try to avoid security procedures that make the system difficult to use. Sometimes you have to accept weaker security to make the system more usable. ↑Log user actions ↑ Maintain a log of user actions that can be analyzed to discover who did what. If users know about such a log, they are less likely to behave in an irresponsible way. ↑Use redundancy and diversity to reduce risk ↑ Keep multiple copies of data and use diverse infrastructure so that an infrastructure vulnerability cannot be the single point of failure.

base decisions an explicit security policy -- ↑ Define a security policy for the organization that sets out the fundamental security requirements that should apply to all organizational systems. ↑Avoid a single point of failure ↑ Ensure that a security failure can only result when there is more than one failure in security procedures. For example, have password and question•based authentication. ↑Fail securely ↑ When systems fail, for whatever reason, ensure that sensitive information cannot be accessed by unauthorized users even although normal security procedures are unavailable.

Chapter Description -- Review reports Records of all design and safety reviews. Team competences Evidence of the competence of all of the team involved in safetyrelated systems development and validation.Process

Construction of a safety argument — ↑Establish the safe exit conditions for a component or a program. ↑Starting from the END of the code, work backwards until you have identified all paths that lead to the exit of the code. ↑Assume that the exit condition is false. ↑Show that, for each path leading to the exit that the assignments made in that path contradict the assumption of an unsafe exit from the component.

Dependable programming guidelines — Security testing and assurance

Design compromises — ↑Adding security features to a system to enhance its security affects other attributes of the system ↑Performance ↑ Additional security checks slow down a system so its response ↑Usability or require additional interactions to complete a transaction. This makes the system less usable and can frustrate system users.

Design decisions from use of COTS — ↑System users authenticated using a name slash password combination. ↑The system architecture is client•server with clients accessing the system through a standard web browser. ↑Information is presented as an editable web form.

Design guidelines for security systems -- engineering SecurityguidelInes BasesecuritydecisionsonanexplicitsecurityIpolicy AvoidasingIepointoffailure Failsecurely BalancesecurityandusabIity LoguseractIons Useredundancyanddiversitytoreducerisk SpecifythefomatofallsysteminputsCompartmentalizeyourassets DesignfordeployIment DesignforrecoverabIity

Design guidelines for security engineering — ↑Design guidelines encapsulate good practice in secure systems design ↑Design guidelines serve two purposes: ↑ They raise awareness of security issues in a software engineering team. Security is considered when design decisions are made. ↑ They can be used as the basis of a review checklist that is applied during the system validation process. ↑Design guidelines here are applicable during software specification and design

Distributed — assets in an equity trading system

Distribution — ↑Distributing assets means that attacks on one system do not necessarily lead to complete loss of system service be different from other platforms so that they do not share a common vulnerability ↑Distribution is particularly important if the risk of denial of service attacks is high

Examples of entries in a security checklist — Security checklist users. computer. attackers to send code strings to the system and then execute them. passwords consist of mixed letters, numbers, and punctuation, and are not normal dictionary entries. They are more difficult to break than simple passwords. specification? Incorrect processing of badly formed inputs is a common cause of security vulnerabilities.

Examples of safety requirements -- SR1: The system shall not deliver a single dose of insulin that is greater than a specified maximum dose for a system user. SR2: The system shall not deliver a daily cumulative dose of insulin that is greater than a specified maximum daily dose for a system user. SR3: The system shall include a hardware diagnostic facility that shall be executed at least four times per hour. SR4: The system shall include an exception handler for all of the exceptions that SR5: The audible alarm shall be sounded when any hardware or software displayed. SR6: In the event of an alarm, insulin delivery shall be suspended until the user has reset the system and cleared the alarm.

Examples of security terminology (Mentcare) — Term Example Asset Therecordsofeachpatientththatisreceivingorhasreceivedtreatment. Exposure Potential financial loss from future patients who do not seek treatment becausetheydonottrusttheclinictomaintaintheirdata.Financialloss fromlegalactionbythespportsstar.Lossofreputation. Vulnerability A weak password system which makes it easy for users to set guessablepasswords.Useridsthatarethesameasnames. Attack Animpersonationofanauthorizeduser. Threat An unauthorized user will gain access to the system by guessing the credentials(loginnameandpassword)ofanauthorizeduser. Control A password checking system that disallows user passwords that are propernamesorwordsthatarenormallyincludedinadictionary.

Fault tree analysis -- ↑Three possible conditions that can lead to delivery of incorrect dose of insulin ↑ Incorrect measurement of blood sugar level ↑ Failure of delivery system ↑ Dose delivered at wrong time ↑By analysis of the fault tree, root causes of these hazards related to software are: ↑ Algorithm error ↑ Arithmetic error

Fault•tree analysis — ↑A deductive top•down technique. ↑Put the risk or hazard at the root of the tree and identify the system states that could lead to that hazard. ↑Where appropriate, link these with 'and' or 'or' conditions. ↑A goal should be to minimise the number of single causes of system failure.

Formal methods cannot guarantee safety — system users. Users rarely understand formal notations so they cannot directly read the formal specification to find errors and omissions, and complex, so, like large and complex programs, they usually contain errors. way that the system is used. If the system is not used as anticipated, then the system's behavior lies outside the scope of the proof.

Formal verification -- ↑Formal methods can be used when a mathematical specification of the system is produced. ↑They are the ultimate static verification technique that process: analyzed for consistency. This helps discover specification errors and omissions. ↑ Formal arguments that a program conforms to its mathematical programming and design errors.

Fundamental security — ↑If a system is a networked system and is insecure then statements about its reliability and its safety are unreliable. ↑These statements depend on the executing system and the developed system being the same. However, intrusion can change the executing system and slash or its data. ↑Therefore, the reliability and safety assurance is no longer valid.

Hazard analysis — ↑Hazard analysis involves identifying hazards and their root causes. ↑There should be clear traceability from identified hazards through their analysis to the actions taken during the process to ensure that these hazards have been covered. the process.

Hazard assessment — ↑Estimate the risk probability and the risk severity. ↑It is not normally possible to do this precisely so relative values are used such as 'unlikely', 'rare', 'very high', etc. ↑The aim must be to exclude risks that are likely to arise or that have high severity.

Hazard identification — hazard: ↑ Physical hazards ↑ Electrical hazards ↑ Biological hazards ↑ Service failure hazards ↑ Etc

Hazard log (2) — Title: System safety design requirements 1. the system shall include self•testing software that will test he system 2. the self•checking s/w shall be executed every so often 3. In the event if a fault in teh system, an audible warning and display shall indicate the fault and its discovery 4. the system shall incorporate an override system taht allows the system user to modify the computed dose of insulin to be delivered 5. the amount of override shall be no greater than a pre•set value (maxOverride)

Hazard-driven analysis -- ↑Hazard identification ↑Hazard assessment ↑Hazard analysis ↑Safety requirements specification

Hazards — ↑Situations or events that can lead to an accident ↑ Stuck valve in reactor control system ↑ Incorrect computation by software in navigation system ↑ Failure to detect possible allergy in medication prescribing system ↑Hazards do not inevitably result in accidents— accident prevention actions can be taken.

Informal safety argument based on — demonstrating contradictions

Insulin dose computation with safety checks -- ●● The insulin dose to be delivered is a function of blood sugar level. ●● the previous dose delivered and the time of delivery of the previous dose currentDose = computeInsulin () ; if (previousDose == 0) { if (currentDose > maxDose slash 2) currentDose = maxDose slash 2 ; } else if (currentDose > (previousDose * 2)) { currentDose < minimumDose) currentDose = 0 ; else if (currentDose > maxDose) currentDose = maxDose ; administerInsulin (currentDose) ;

Insulin pump - software risks — ↑Arithmetic error ↑ A computation causes the value of a variable to overflow or underflow. ↑ Maybe include an exception handler for each type of arithmetic error. ↑Algorithmic error ↑ Compare dose to be delivered with previous dose or safe maximum doses. Reduce dose if too high.

Insulin pump risks — ↑Insulin overdose (service failure). ↑Insulin underdose (service failure). ↑Power failure due to exhausted battery (electrical). ↑Electrical interference with other medical equipment (electrical). ↑Poor sensor and actuator contact (physical). ↑Parts of machine break off in body (physical). ↑Infection caused by introduction of machine (biological). ↑Allergic reaction to materials or insulin (biological).

Insulin pump safety argument — Arguments are based on claims and evidence. ↓Insulin pump safety: ↑ Claim: The maximum single dose of insulin to be delivered (CurrentDose) will not exceed MaxDose. ↑ Evidence: Safety argument for insulin pump (discussed later) ↑ Evidence: Test data for insulin pump. The value of currentDose ↑ Evidence: Static analysis report for insulin pump software revealed no anomalies that affected the value of CurrentDose ↑ Argument: The evidence presented demonstrates that the maximum dose of insulin that can be computed = MaxDose.

Key points — ↑key issues when designing a secure systems architecture include organizing the system structure to protect key assets and distributing the system assets to minimize the losses from a successful attack. ↑Security design guidelines sensitize system designers to provide a basis for creating security review checklists. ↑Security validation is difficult because security requirements state what should not happen in a system, rather than what should. Furthermore, system attackers weaknesses than is available for security testing.

Levels of static analysis — ↑Characteristic error checking ↑ The static analyzer can check for patterns in the code that are characteristic of errors made by programmers using a particular language. ↑User•defined error checking ↑ Users of a programming language define error patterns, thus extending the types of error that can be detected. This allows specific rules that apply to a program to be checked. ↑Assertion checking ↑ Developers include formal assertions in their program and relationships that must hold. The static analyzer symbolically executes the code and highlights potential problems.

Normal accidents — ↑Accidents in complex systems rarely have a single cause as these systems are designed to be resilient to a single point of failure ↑ Designing systems so that a single point of failure does not cause an accident is a fundamental principle of safe systems design. ↑Almost all accidents are a result of combinations of malfunctions rather than single failures. ↑It is probably the case that anticipating all problem combinations, especially, in software controlled systems is impossible so achieving complete safety is impossible. Accidents are inevitable.

Operational risk assessment — ↑This risk assessment process focuses on the use of the system and the possible risks that can arise from human behavior. ↑Operational risk assessment should continue after a system has been installed to take account of how the system is used. used in different ways from those originally planned. These changes lead to new security requirements that have to be implemented as the system evolves.

Operational security — ↑Primarily a human and social issue ↑Concerned with ensuring the people do not take actions ↑ E.g. Tell others passwords, leave computers logged on ↑Users sometimes take insecure actions to make it easier for them to do their jobs ↑There is therefore a trade•off between system security and system effectiveness.

Organizational security policies — ↑Security policies should set out general information access strategies that should apply across the organization. ↑The point of security policies is to inform everyone in an organization about security so these should not be long and detailed technical documents. ↑From a security engineering perspective, the security policy defines, in broad terms, the security goals of the organization. ↑The security engineering process is concerned with implementing these goals.

Preliminary risk assessment — ↑The aim of this initial risk assessment is to identify generic risks that are applicable to the system and to decide if an adequate level of security can be achieved at a reasonable cost. ↑The risk assessment should focus on the identification and analysis of high•level risks to the system. ↑The outcomes of the risk assessment process are used to help identify security requirements.

Processes for safety assurance — ↑Process assurance is important for safety•critical systems development. ↑ Safety requirements are sometimes 'shall not' requirements so cannot be demonstrated through testing. software process that record the analyses that have been carried out and the people responsible for these. to subsequent legal actions.

Program paths — ↑ Can only happen if CurrentDose is >= minimumDose and <= maxDose. ↑ currentDose = 0. ↑ currentDose = maxDose. ↑In all cases, the post conditions contradict the unsafe condition that the dose administered is greater than maxDose.

Protection requirements -- knowledge of information representation and system distribution ↑Separating patient and treatment information limits the amount of information (personal patient data) that needs to be protected ↑Maintaining copies of records on a local client protects against denial of service attacks on the server

Protection — ↑Platform•level protection ↑ Top•level controls on the platform on which a system runs. ↑Application•level protection ↑ Specific protection mechanisms built into the application itself e.g. additional password protection. ↑Record•level protection ↑ Protection that is invoked when access to specific information is requested

↓These lead to a layered protection architecture

QA Records of the quality assurance processes (see Chapter 24) -- carried out during system development. 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. Associated safety cases Structured arguments ↑Safety cases should be based around structured arguments that present evidence to justify the assertions made in these arguments. ↑The argument justifies why a claim about system safety and security is justified by the available evidence.

Regulation -- processes have been used in system development ↑For example: ↑ The specification of the system that has been developed and records of the checks made on that specification. ↑ Evidence of the verification and validation processes that have been carried out and the results of the system verification and validation. ↑ Evidence that the organizations developing the system have defined and dependable software processes that include safety assurance reviews. There must also be records that show that these processes have been properly enacted.

Risk classification for the insulin pump -- Identified hazard Hazard probability Accident severity Estimated risk Acceptability 1.Insulin overdose computation Medium High High Intolerable computation Medium Low Low Acceptable hardware monitoring system Medium Medium Low ALARP incorrectly fitted High High High Intolerable patient Low High Medium ALARP infection Medium Medium Medium ALARP interference Low High Medium ALARP

Risk reduction -- ↑The aim of this process is to identify dependability requirements that specify how the risks should be managed and ensure that accidents slash incidents do not arise. ↑Risk reduction strategies ↑ Hazard avoidance; ↑ Hazard detection and removal; ↑ Damage limitation.

Safety achievement — ↑Hazard avoidance ↑ The system is designed so that some classes of hazard simply cannot arise. ↑Hazard detection and removal ↑ The system is designed so that hazards are detected and removed before they result in an accident. ↑Damage limitation ↑ The system includes protection features that minimise the

Safety and dependability cases — ↑Safety and dependability cases are structured documents that set out detailed arguments and evidence that a required level of safety or dependability has been achieved. ↑They are normally required by regulators before a system can be certified for operational use. The regulator's responsibility is to check that a system is as safe or dependable as is practical. ↑Regulators and developers work together and negotiate what needs to be included in a system safety slash dependability case.

Safety and reliability -- ↑Safety and reliability are related but distinct. ↑ In general, reliability and availability are necessary but not sufficient conditions for system safety ↑Reliability is concerned with conformance to a given specification and delivery of service ↑Safety is concerned with ensuring system cannot cause damage irrespective of whether or not it conforms to its specification. ↓ System reliability is essential for safety but is not enough ↑ Reliable systems can be unsafe

Safety assurance processes — ↑Process assurance involves defining a dependable process and ensuring that this process is followed during the system development. ↑Process assurance focuses on: ↑ Do we have the right processes? Are the processes appropriate for the level of dependability required. Should include requirements management, change management, reviews and inspections, etc. ↑ Are we doing the processes right? Have these processes been followed by the development team. ↑Process assurance generates documentation ↑ Agile processes therefore are rarely used for critical systems.

Safety critical systems — ↑Systems where it is essential that system operation is always safe i.e. the system should never cause damage to people or the system's environment ↑Examples ↑ Control and monitoring systems in aircraft ↑ Process control systems in chemical manufacture ↑ Automobile control systems such as braking and engine management systems

Safety criticality -- ↑Primary safety•critical systems ↑ Embedded software systems whose failure can cause the associated hardware to fail and directly threaten people. Example is the insulin pump control system. ↑Secondary safety•critical systems ↑ Systems whose failure results in faults in other (socio•technical) systems, which can then have safety consequences, lead to inappropriate treatment being prescribed. • Infrastructure control systems are also secondary safety•critical systems.

Safety engineering processes — ↑Safety engineering processes are based on reliability engineering processes ↑ Plan•based approach with reviews and checks at each stage in the process ↑ General goal of fault avoidance and fault detection ↑ Must also include safety reviews and explicit identification and tracking of hazards

Safety related process activities — ↑Creation of a hazard logging and monitoring system. ↑Appointment of project safety engineers who have explicit responsibility for system safety. ↑Extensive use of safety reviews. ↑Creation of a safety certification system where the safety of critical components is formally certified.

Safety specification — ↑The goal of safety requirements engineering is to identify protection requirements that ensure that system failures do not cause injury or death or environmental damage. they define situations and events that should never occur. ↑Functional safety requirements define: ↑ Checking and recovery features that should be included in a system ↑ Features that provide protection against system failures and external attacks

Safety terminology — Term Definition 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 is an example of an accident. Hazard A condition with the potential for causing or contributing to an accident. A failure of the sensor that measures blood glucose is an example of a hazard. Damage A measure of the loss resulting from a mishap. Damage can range from many people being killed as a result of an accident to minor injury or property damage. Damage resulting from an overdose of insulin could be serious injury or the death of the user of the insulin pump. Hazard severity An assessment of the worst possible damage that could result from a particular hazard. Hazard severity can range from catastrophic, where many people are killed, to minor, where only minor damage results. When an individual death is a possibility, a reasonable assessment of hazard severity is 'very high'. Hazard probability The probability of the events occurring which create a hazard. Probability values tend to 'implausible' (no conceivable situations are likely in which the hazard could occur). The probability of a sensor failure in the insulin pump that results in an overdose is probably low. Risk This is a measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the hazard severity, and the probability that the hazard will lead to an accident. The risk of an insulin overdose is probably medium to low.

Safety — ↑Safety is a property of a system that reflects the system's ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system's environment. ↑It is important to consider software safety as most devices whose failure is critical now incorporate software•based control systems.

Safety2 — Safety is a property of a system that reflects the system's ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system's environment. It is important to consider software safety as most devices whose failure is critical now incorporate software•based control systems.

Secure systems design — ↑Security should be designed into a system–it is very difficult to make an insecure system secure after it has been designed or implemented ↑Architectural design ↑ how do architectural design decisions affect the security of a system? ↑Good practice ↑ what is accepted good practice when designing secure systems?

Security and dependability — ↑Security and safety ↑ An attack that corrupts the system or its data means that analysing the source code of safety critical software and assume the executing code is a completely accurate translation of that induced and the safety case made for the software is invalid. ↑Security and resilience ↑ Resilience is a system characteristic that reflects its ability to resist and recover from damaging events. The most probable damaging event on networked software systems is a cyberattack of some kind so most of the work now done in resilience is aimed at deterring, detecting and recovering from such attacks.

Security assurance -- ↑Vulnerability avoidance ↑ The system is designed so that vulnerabilities do not occur. For example, if there is no external network connection then external attack is impossible ↑Attack detection and elimination ↑ The system is designed so that attacks on vulnerabilities are detected and neutralised before they result in an exposure. For example, virus checkers find and remove viruses before they infect a system ↑Exposure limitation and recovery ↑ The system is designed so that the adverse consequences of a successful attack are minimised. For example, a backup policy allows damaged information to be restored

Security dimensions — ↑Confidentiality people or programs that are not authorized to have access to that information. ↑Integrity unusual or unreliable. ↑Availability be possible.

Security engineering -- ↑Tools, techniques and methods to support the development and maintenance of systems that can resist malicious attacks that are intended to damage a computer•based system or its data. ↑A sub•field of the broader field of computer security.

Security is a business issue — ↑Security is expensive and it is important that security decisions are made in a cost•effective way ↑ There is no point in spending more than the value of an asset to keep that asset secure. ↑Organizations use a risk•based approach to support security decision making and should have a defined security policy based on security risk analysis ↑Security risk analysis is a business rather than a technical process

Security levels — ↑Infrastructure security, which is concerned with maintaining the security of all systems and networks that provide an infrastructure and a set of shared services to the organization. ↑Application security, which is concerned with the security of individual application systems or related groups of systems. ↑Operational security, which is concerned with the secure operation and use of the organization's systems.

Security policies — ↑The responsibilities of individual users, managers and the organization ↑ The security policy should set out what is expected of users e.g. strong passwords, log out of computers, office security, etc. ↑Existing security procedures and technologies that should be maintained continue to use existing approaches to security even where these have known limitations.

Security requirement classification — ↑Risk avoidance requirements set out the risks that should be avoided by designing the system so that these risks simply cannot arise. ↑Risk detection requirements define mechanisms that identify the risk if it arises and neutralise the risk before losses occur. ↑Risk mitigation requirements set out how the system should be designed so that it can recover from and restore system assets after some loss has occurred.

Security requirements — ↑A password checker shall be made available and shall be run daily. Weak passwords shall be reported to system administrators. ↑Access to the system shall only be allowed by approved client computers. ↑All client computers shall have a single, approved web browser installed by system administrators.

Security risk assessment and management -- ↑Risk assessment and management is concerned with assessing the possible losses that might ensue from attacks on the system and balancing these losses these losses. ↑Risk management should be driven by an organisational security policy. ↑Risk management involves ↑ Preliminary risk assessment ↑ Life cycle risk assessment ↑ Operational risk assessment

Security risk assessment -- ↑Attack assessment ↑ Decompose threats into possible attacks on the system and the ↑Control identification asset. ↑Feasibility assessment ↑ Assess the technical feasibility and cost of the controls. ↑Security requirements definition ↑ Define system security requirements. These can be infrastructure or application system requirements.

Security specification — ↑ Security specification has something in common with safety requirements specification –in both cases, your concern is to avoid something bad happening. ↑ Four major differences ↑ Safety problems are accidental –the software is not operating in a hostile environment. In security, you must assume that attackers have knowledge of system weaknesses ↑ When safety failures occur, you can look for the root cause or weakness that led to the failure. When failure results from a deliberate attack, the ↑ Shutting down a system can avoid a safety•related failure. Causing a ↑ Safety•related events are not generated from an intelligent adversary. An attacker can probe defenses over time to discover weaknesses.

Security terminology — Term Definition systemitselfordatausedbythatsystem. Attack An exploitation of a system's vulnerability. Generally, this is from outside the systemandisadeliberateattempttocausesomedamage. 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 Exposure Possible loss or harm to a computing system. This can be loss or damage to data,orcanbealossoftimeandeffortifrecoveryisnecessaryafterasecurity breach. Threat Circumstancesthathavepotentialtocauselossorharm.Youcanthinkofthese asasystemvulnerabilitythatissubjectedtoanattack. Vulnerability Aweaknessinacomputer•basedsystemthatmaybeexploitedtocauselossor harm.

Security validation -- ↑Experience•based testing ↑ The system is reviewed and analysed against the types of attack that are known to the validation team. ↑Penetration testing ↑ A team is established whose goal is to breach the security of the system by simulating attacks on the system. ↑Tool•based analysis ↑ Various security tools such as password checkers are used to analyse the system in operation. ↓Formal verification ↑ The system is verified against a formal security specification.

Security — ↑The security of a system is a system property that reflects the system's ability to protect itself from accidental or deliberate external attack. ↑Security isessential as most systems are networked so that external access to the system through the Internet is possible. ↑Security is an essential pre•requisite for availability, reliability and safety.

Securitytesting — ↑Testing the extent to which the system can protect itself from external attacks. ↑ Problems with security testing ↑ Security requirements are 'shall not' requirements i.e. they specify what should not happen. It is not usually possible to define security requirements as simple constraints that can be checked by the system. ↑ The people attacking a system are intelligent and look for vulnerabilities. They can experiment to discover weaknesses and loopholes in the system.

Social acceptability of risk — ↑The acceptability of a risk is determined by human, social and political considerations. ↑In most societies, the boundaries between the regions are pushed upwards with time i.e. society is less willing to accept risk ↑Risk assessment is subjective ↑ Risks are identified as probable, unlikely, etc. This depends on who is making the assessment.

Software in safety-critical systems — decisions made by the software and subsequent actions are safety•critical. Therefore, the software behaviour is directly related to the overall safety of the system. ↑Software is extensively used for checking and monitoring other safety•critical components in a system. For example, all aircraft engine components are monitored by software looking for early indications of component failure. This software is safety•critical because, if it fails,

Software safety arguments — ↑Safety arguments are intended to show that the system cannot reach in 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 ↑ Assume that an unsafe state can be reached; ↑ Show that this is contradicted by the program code, developed.

Software safety benefits — ↑Although software failures can be safety•critical, the use of software control systems contributes to increased system safety ↑ Software monitoring and control allows a wider range of conditions to be monitored and controlled than is possible using electro•mechanical safety systems. ↑ Software control allows safety strategies to be adopted that reduce the amount of time people spend in hazardous environments. ↑ Software can detect and correct safety•critical operator errors.

Specify the format of all system inputs — ↑ If input formats are known then you can check that all inputs are within range so that unexpected inputs don't cause problems. ↑Compartmentalize your assets ↑ Organize the system so that assets are in separate areas and users only have access to the information that they need rather than all system information. ↓Design for deployment ↑ Design the system to avoid deployment problems ↓Design for recoverability ↑ Design the system to simplify recoverability after a successful attack.

Static program analysis — ↑Static analysers are software tools for source text processing. ↑They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team. ↑They are very effective as an aid to inspections ● they are a supplement to but not a replacement for inspections.

Strategy use -- ↑Normally, in critical systems, a mix of risk reduction strategies are used. ↑In a chemical plant control system, the system will include sensors to detect and correct excess pressure in the reactor. ↓However, it will also include an independent protection system that opens a relief valve if dangerously high pressure is detected.

Structured safety arguments — ↑Structured arguments that demonstrate that a system meets its safety obligations. ↑It is not necessary to demonstrate that the program works as intended; the aim is simply to demonstrate safety. ↓Generally based on a claim hierarchy. ↑ You start at the leaves of the hierarchy and demonstrate safety. This implies the higher•level claims are true.

System safety design requirements — clock, and the insulin delivery system. components, an audible warning shall be issued and the pump display shall indicate the name of the component where the fault has been discovered. The delivery of insulin shall be suspended. the computed dose of insulin that is to be delivered by the system. which is set when the system is configured by medical staff. Safety reviews ↓Driven by the hazard register. ↓For each identified hazard, the review team should assess the system and judge whether or not the system can cope with that hazard in a safe way.