

# Guidelines for Safehome Implementation

Moonzoo Kim

CS Division of EECS Dept.

KAIST

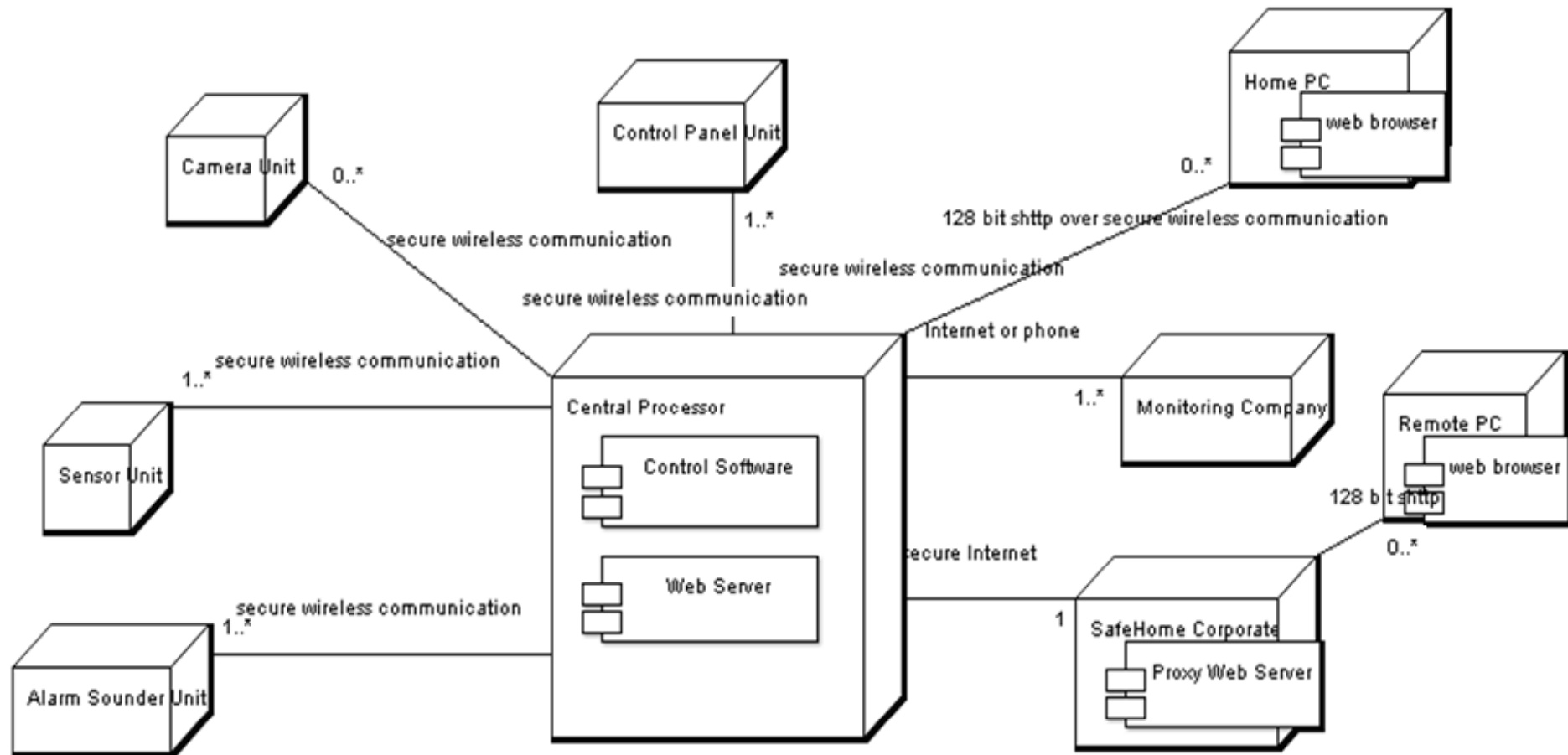
[moonzoo@cs.kaist.ac.kr](mailto:moonzoo@cs.kaist.ac.kr)

<http://pswlab.kaist.ac.kr/courses/CS350-07>

# Overview of the Guidelines

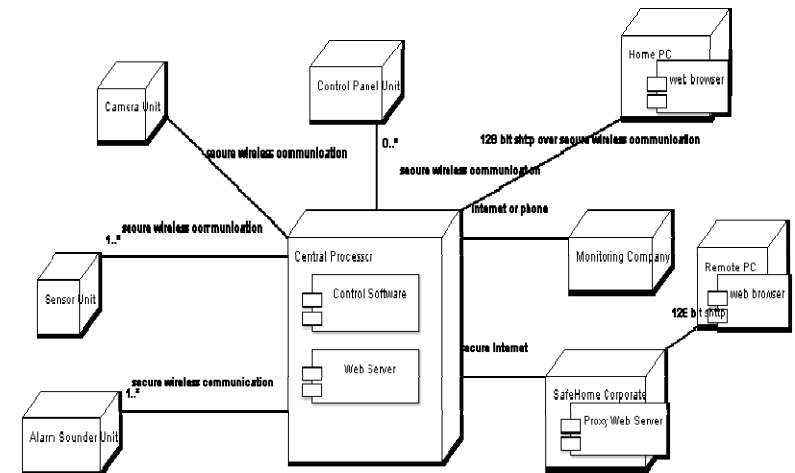
- Deployment diagrams
  - Original one v.s. simplified one
  - Alternative implementation solutions
- Virtual devices
  - Screen shots
  - APIs
- Javadoc
- Junit
- To-do-list

# Original Deployment Diagram (1/2)

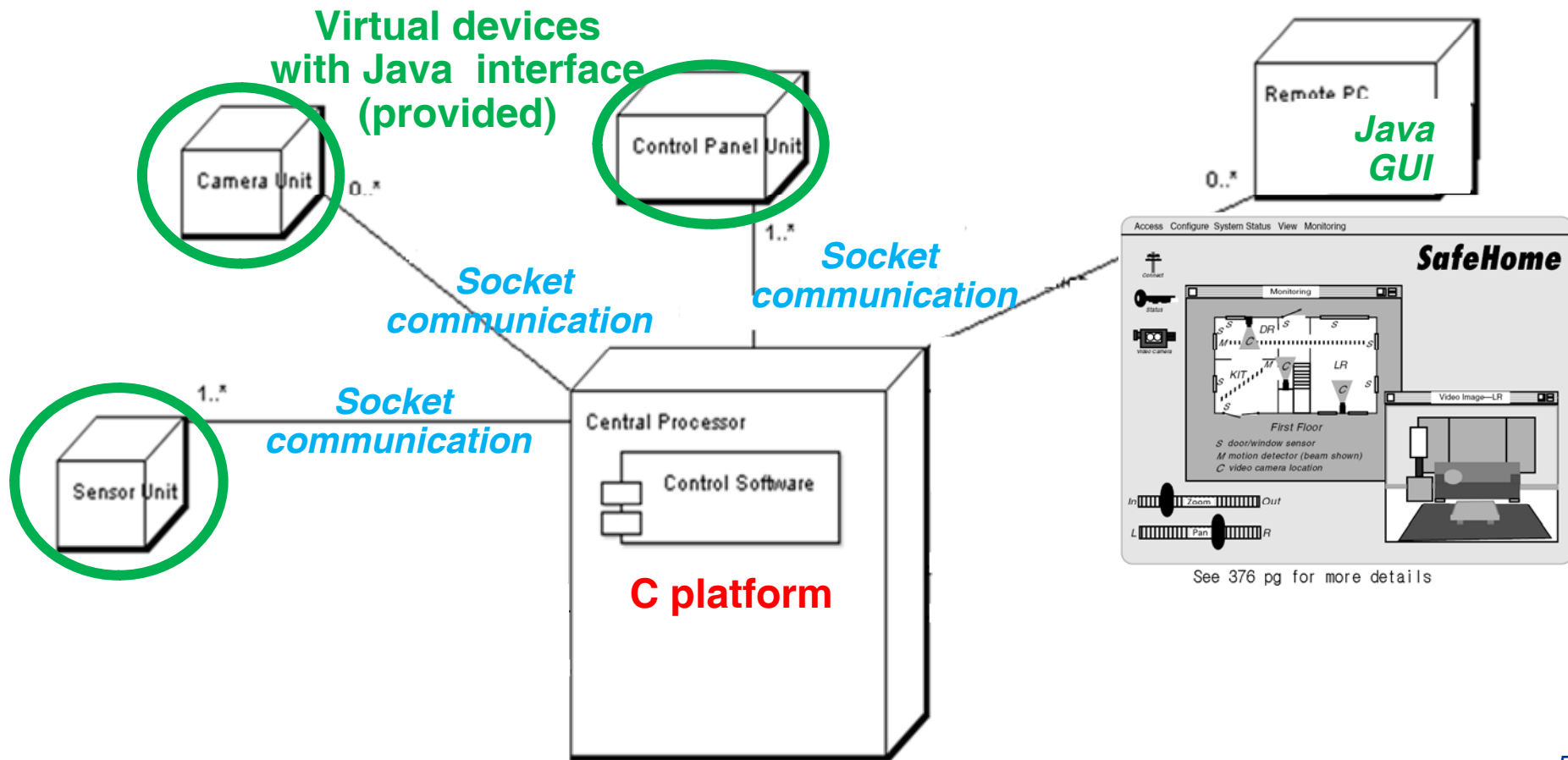


# Original Deployment Diagram (2/2)

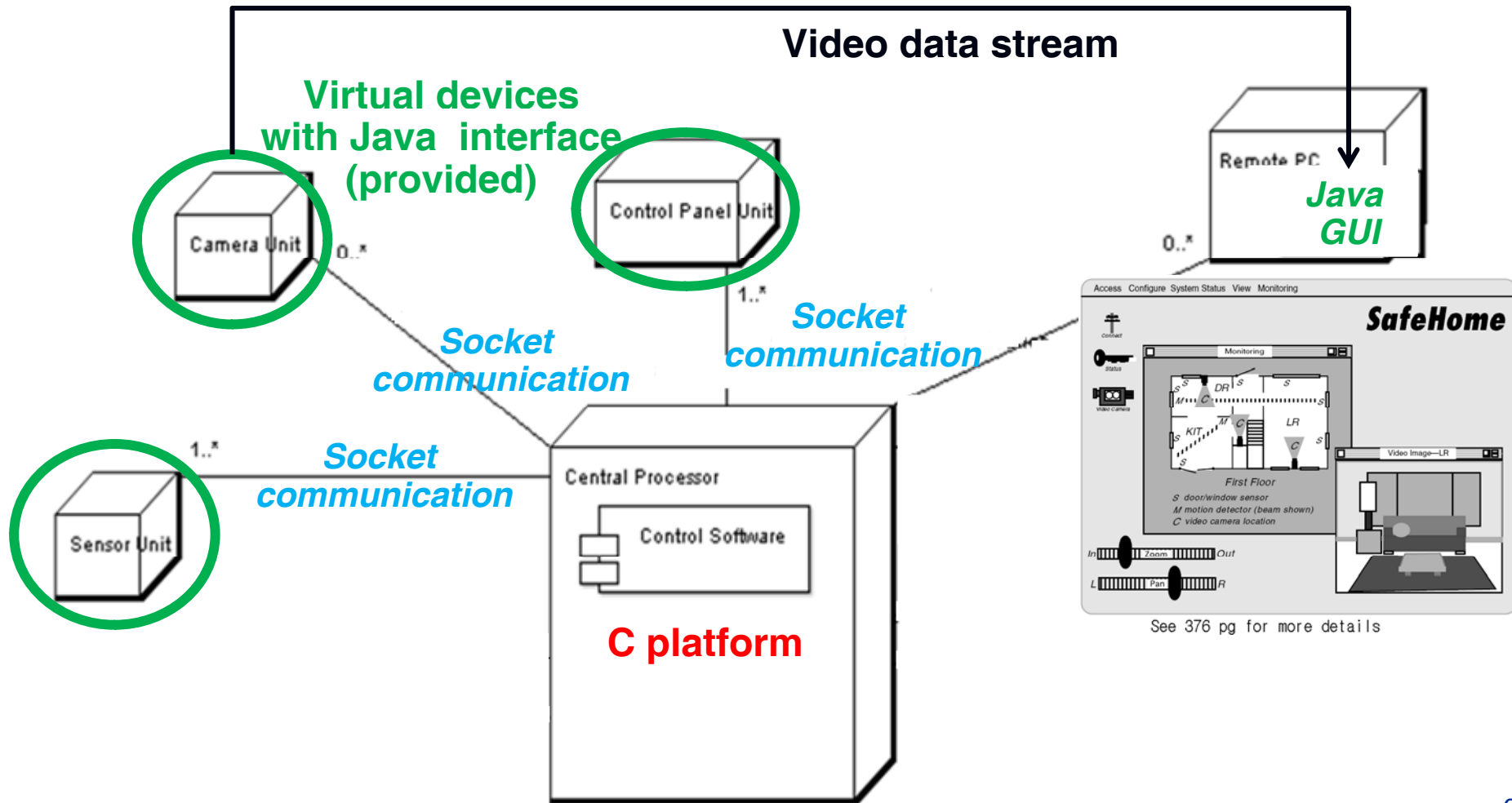
- Safehome is a safety critical embedded system
  - The central processor (CP) should be hosted by a **dedicated reliable HW/SW platform** for non-stop operations
  - Java is **not** suitable for this purpose
    - Java is heavy compared to C/C++
    - JVM is unnecessarily complex platform for Safehome
      - Core functions of Safehome are simple
- Mind the **heterogeneous** platforms
  - Safehome central processor communicates with diverse devices through wireless network connections
  - **Flexible communication** among the components is crucial



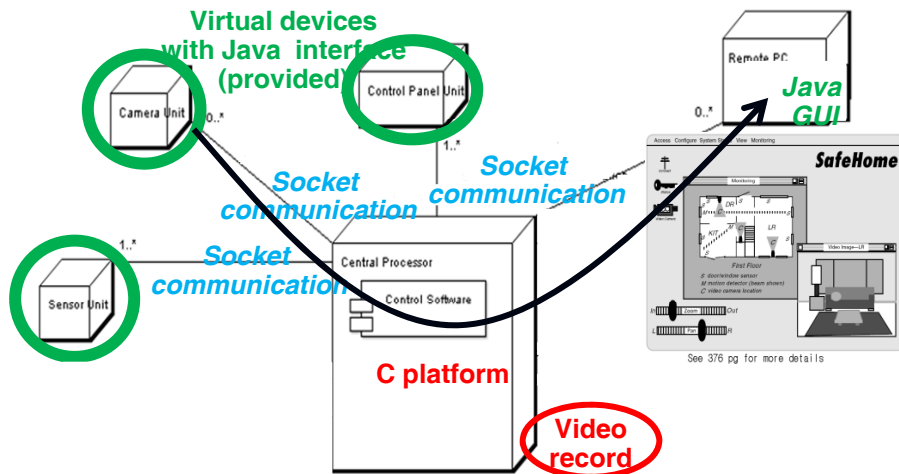
# Simplified Deployment Diagram



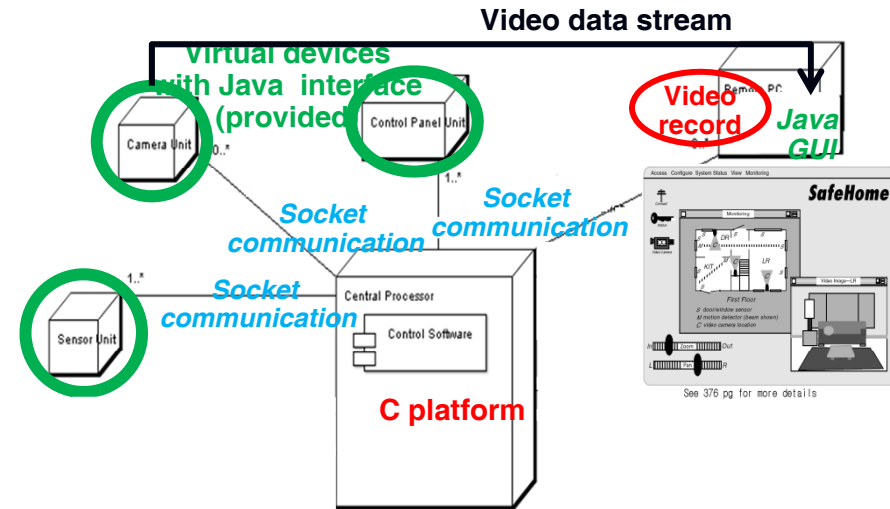
# Alternative Implementation Choice



# Comparison

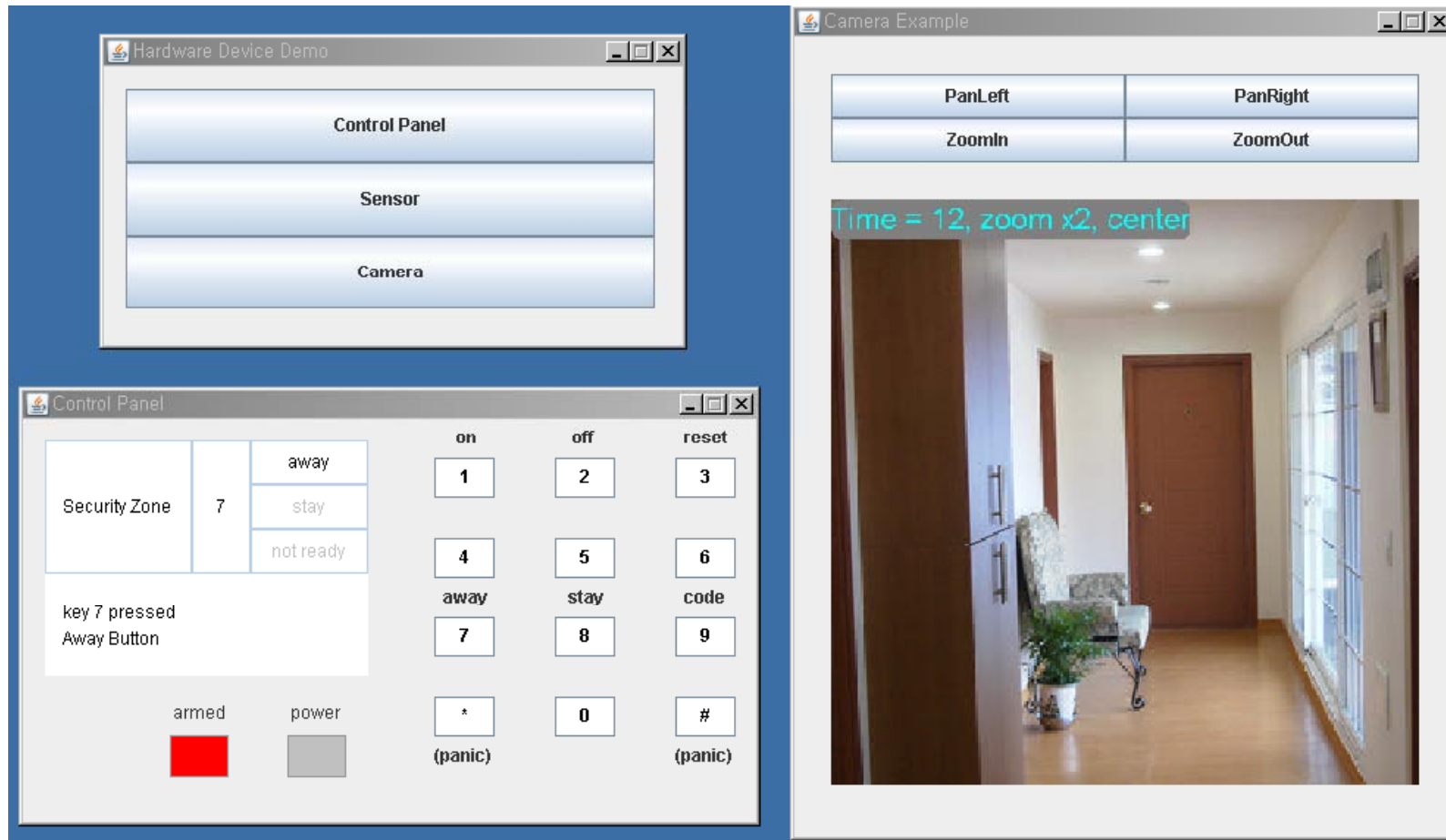


- If video recording is considered crucial, video stream should be handled by CP
  - Pros: reliable video recording
  - Cons: unnecessary computational resource of CP consumed for transferring video stream to the GUI



- If video recording is considered optional, video stream can bypass to the home PC without going through the central processor
  - Pros: saved network bandwidth and processing power of the CP
  - Cons: additional network connections and unreliable video recording

# Screenshot of Virtual Devices (1/2)





# Screenshot of Virtual Devices (2/2)

The screenshot shows a software interface for virtual devices. It consists of three main windows:

- Hardware Device Demo:** A window with three buttons: "Control Panel", "Sensor", and "Camera".
- Sensor Test:** A window with two tables. The first table is for WinDoor sensors, and the second is for Motion sensors. Both tables show ID ranges and input IDs, along with "open/close" and "detect/clear" buttons.
- Sensor Status:** A window displaying a table of sensor types and IDs. The table has three columns: "Type and ID", "Test", and "Read". The "Test" column contains "enable" for all entries. The "Read" column contains "close", "open", "close", "close", "close", "clear", "clear", "detect", "clear", "clear". Below the table are three buttons: "refresh", "Enable All", and "Disable All".

| Type and ID | Test   | Read   |
|-------------|--------|--------|
| WinDoor 1   | enable | close  |
| WinDoor 2   | enable | open   |
| WinDoor 3   | enable | close  |
| WinDoor 4   | enable | close  |
| WinDoor 5   | enable | close  |
| Motion 1    | enable | clear  |
| Motion 2    | enable | clear  |
| Motion 3    | enable | detect |
| Motion 4    | enable | clear  |
| Motion 5    | enable | clear  |

# Virtual Device APIs

- safehome package contains `MainDemo.java` which is a main file for the demo application
- `safehome.device` contains virtual devices and its **interfaces**. You may want to inherit those device classes for your purpose
- You should read carefully the following three java files
  - `safehome.MainDemo.java`
  - `safehome.device.interfaceCamera.java`
  - `safehome.device.interfaceSensor.java`
- All other java files should not be modified unless you are for your

# Virtual Device APIs for Control Panel

- Callback functions of the control panel's buttons
  - A call back function is invoked when the corresponding button of the control panel GUI is pressed.
  - You can inherit the super class  
`safehome.device.DeviceControlPanelAbstract`
- When you push a button <n> of the control panel, the corresponding call-back function `abstract public void button<n>()` is invoked.
  - You can inherit that method as shown in `MainDemo.java`
- Output to the control panel
  - `public void setDisplayAway(boolean on)`
  - `public void setDisplayStay(boolean on)`
  - `public void setDisplayNotReady(boolean on)`
  - `public void setDisplayShortMessage(String message)`
  - `public void setLedArmed(boolean on)`

# Virtual Device APIs for Camera

- Note that we do **not** implement video recording function for saving your effort 😊
- There are four functions to control a camera
  - `public boolean panRight();`
  - `public boolean panLeft();`
  - `public boolean zoomIn();`
  - `public boolean zoomOut();`
- Image acquisition is performed by
  - `public BufferedImage getView();`
  - You may serialize the obtained `BufferedImage` object and transfer the serialized image to the central processor. Then the central processor simply deliver the serialized image to the GUI

# Virtual Device APIs for Sensors

- `public int getID`
- `public boolean read()`
- `public void enable()`
- `public void disable()`

All Classes

Packages

[safehome](#)

[safehome.device](#)

All Classes

[CameraTest](#)

[CameraView](#)

[ControlPanel](#)

[DeviceCamera](#)

[DeviceControlPanel/](#)

[DeviceMotionDetect](#)

[DeviceSensorTester](#)

[DeviceWinDoorSens](#)

[interfaceCamera](#)

[interfaceSensor](#)

[MainDemo](#)

[SafeHomeSensorTe](#)

[SensorTest](#)

Overview Package **Class** Use Tree Deprecated Index Help

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

safehome

## Class ControlPanel

[java.lang.Object](#)

└ [java.awt.Component](#)

└ [java.awt.Container](#)

└ [java.awt.Window](#)

└ [java.awt.Frame](#)

└ [javax.swing.JFrame](#)

└ [safehome.device.DeviceControlPanelAbstract](#)

└ **safehome.ControlPanel**

All Implemented Interfaces:

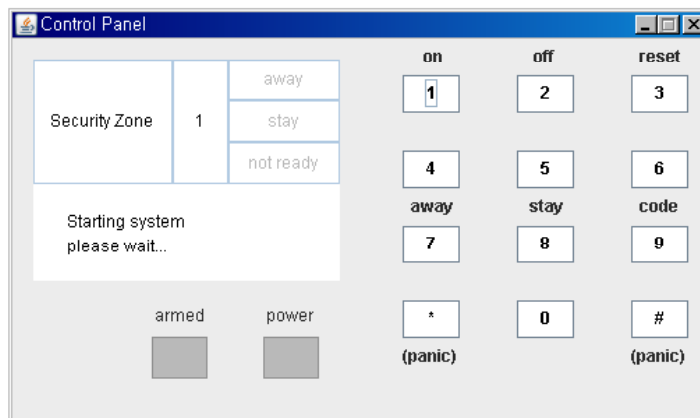
[ActionListener](#), [ImageObserver](#), [MenuContainer](#), [Serializable](#),  
[EventListener](#), [Accessible](#), [RootPaneContainer](#), [WindowConstants](#)

class **ControlPanel**

extends [DeviceControlPanelAbstract](#)

This class is a simple implementation of the abstract class [DeviceControlPanelAbstract](#). All abstract methods are implemented simply to show that each button is working well.

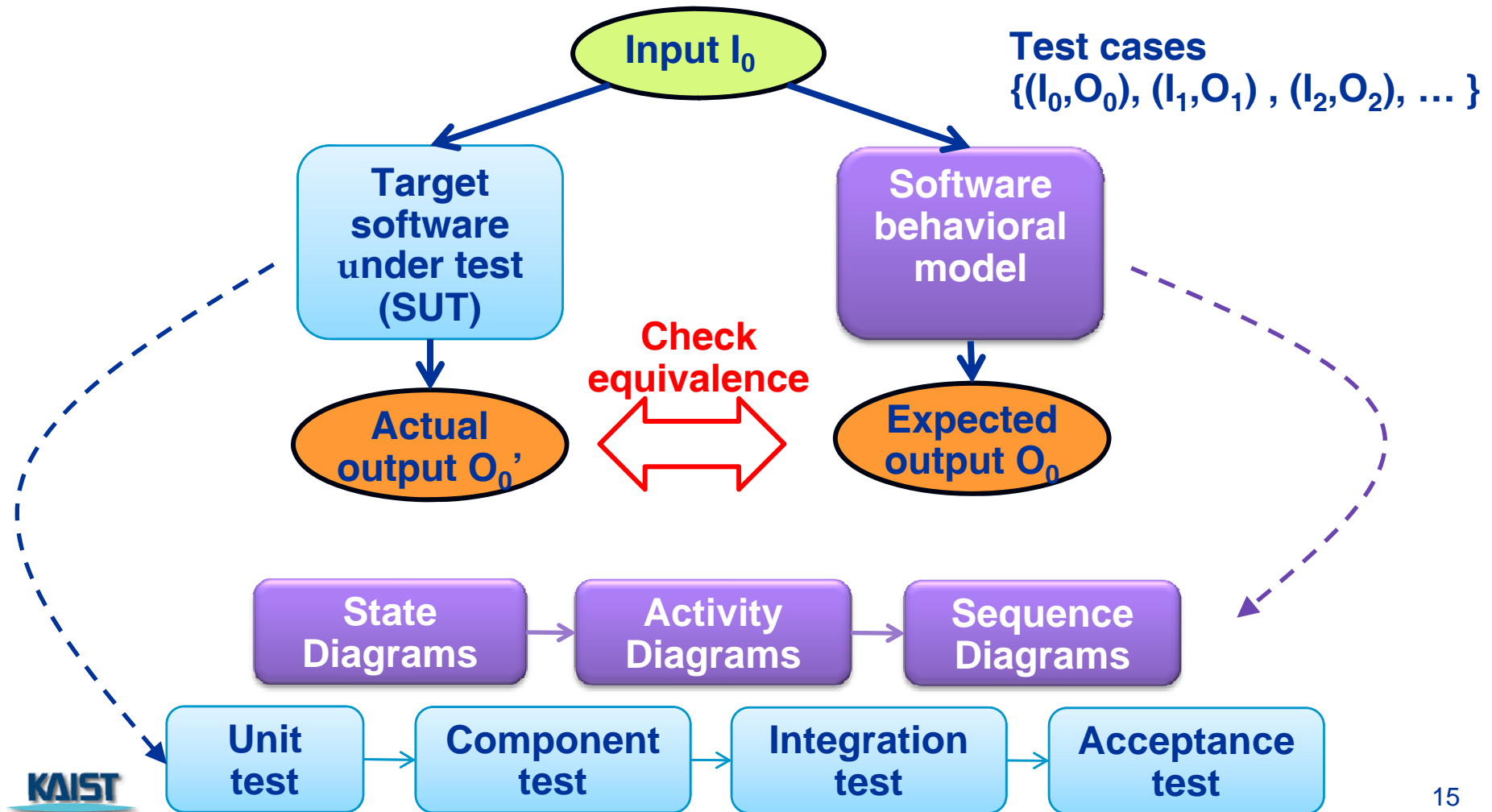
The control panel consists of three parts. It looks like this



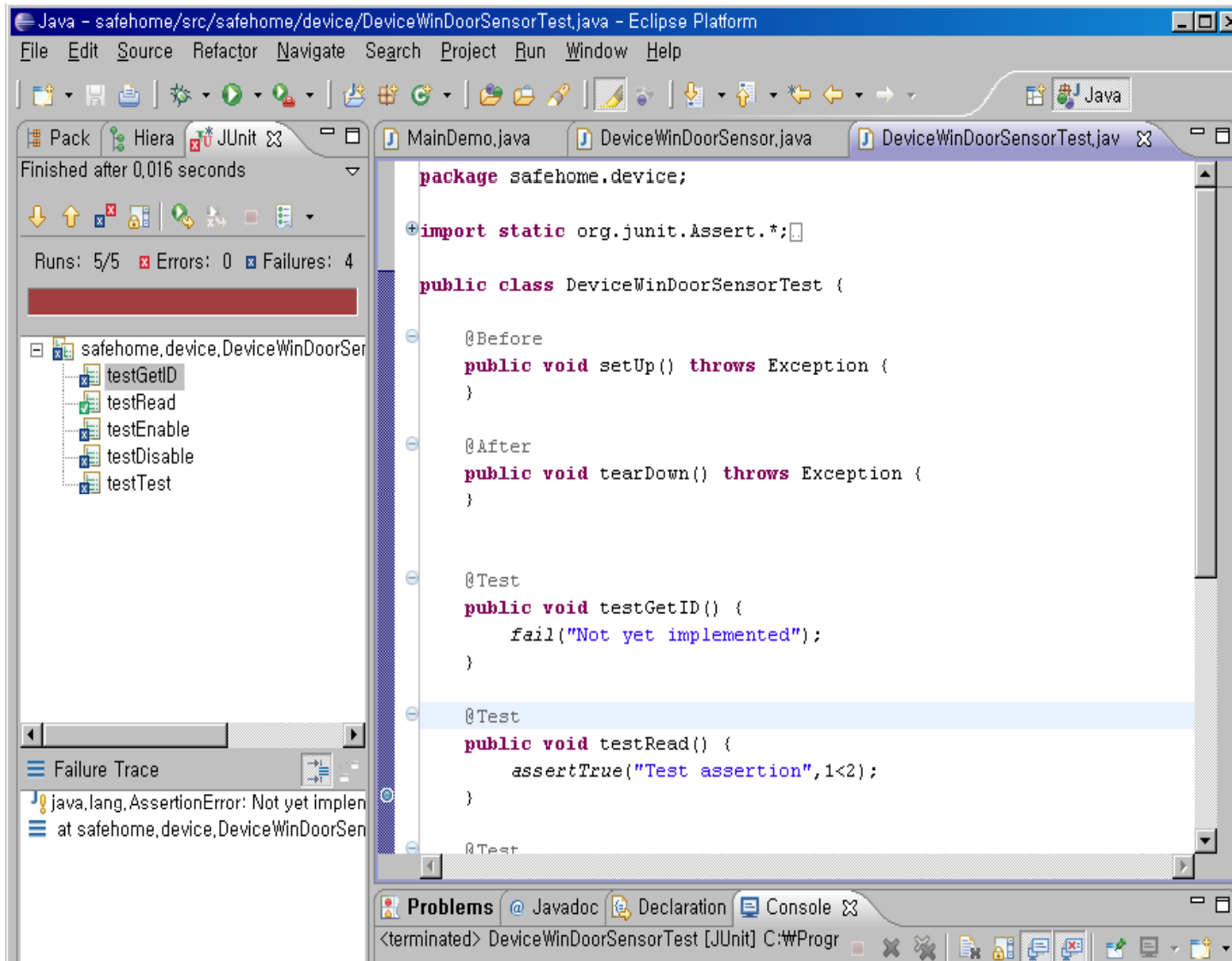
# Javadoc Example

```
/**
 * This class is a simple implementation of the abstract
 * class <code>{@link DeviceControlPanelAbstract}</code>. All abstract methods are
 * implemented simply to show that each button is working well.
 * <p>
 * The control panel consists of three parts. It looks like this
 * <p>
 * <img src = "doc-files/initial.PNG" />
 * <p>
 * LCD on the left-up shows the current status, which button is pressed,
 * the system is in stay or away mode, and the system is ready or not.
 * Two LEDs on the left-down indicate the system is armed or not and
 * the system is power-on or off. The keypad on the right-side has 12 buttons,
 * 0 ~ 9, *, #. The corresponding action occurs when a user presses each button.
 *
 *
 * @author cs350 TA
 * @see safehome.device.DeviceControlPanelAbstract
 */
class ControlPanel extends DeviceControlPanelAbstract
{
    /**
     * Constructs a control panel frame.
     * <p>
     * Pre-condition: None.
     * <p>
     * Post-condition: The control panel frame is ready. It is initially invisible.
     * @see safehome.device.DeviceControlPanelAbstract#DeviceControlPanelAbstract()
     */
    public ControlPanel()
    {
        super();
        setDisplayShortMessage1("Starting system");
        setDisplayShortMessage2("please wait...");
    }
}
```

# Model-based Testing



# JUnit in Eclipse for Testing





# To Do List (1/2)

- Develop software components for devices based on the provided virtual devices in Java and CP in C.
  - You will need network programming to make communication between the devices and CP
  - Make detailed comments as much as possible using JavaDoc
    - Your C code should have rich comments as well
- Perform quality testing
  - Document your test plan and test cases in detail.
    - Provide rationale why selected test cases are good enough
  - You may use advanced testing techniques such as Java Modeling Language (JML), AspectJ or C-BMC (C-bounded model checker)
  - Also, describe bug/revision history
    - See linux kernel changelog as examples

# Example of Linux Kernel ChangeLog-2.6.7

|                        |              |                   |
|------------------------|--------------|-------------------|
| ChangeLog-2.6.0        | 13KB         | 2003-12-18        |
| ChangeLog-2.6.1        | 190KB        | 2004-01-09        |
| ChangeLog-2.6.2        | 287KB        | 2004-02-04        |
| ChangeLog-2.6.3        | 301KB        | 2004-02-18        |
| ChangeLog-2.6.4        | 322KB        | 2004-03-11        |
| ChangeLog-2.6.5        | 358KB        | 2004-04-04        |
| ChangeLog-2.6.6        | 487KB        | 2004-05-10        |
| <b>ChangeLog-2.6.7</b> | <b>761KB</b> | <b>2004-06-16</b> |
| ChangeLog-2.6.8        | 883KB        | 2004-08-14        |
| ChangeLog-2.6.8.1      | 1KB          | 2004-08-14        |
| ChangeLog-2.6.9        | 1,264KB      | 2004-10-20        |
| ChangeLog-2.6.10       | 1,517KB      | 2004-12-25        |
| ChangeLog-2.6.11       | 1,461KB      | 2005-03-03        |
| ChangeLog-2.6.11.1     | 2KB          | 2005-03-09        |
| ChangeLog-2.6.11.2     | 1KB          | 2005-03-09        |
| ChangeLog-2.6.11.3     | 8KB          | 2005-03-13        |
| ChangeLog-2.6.11.4     | 1KB          | 2005-03-16        |
| ChangeLog-2.6.11.5     | 6KB          | 2005-03-19        |
| ChangeLog-2.6.11.6     | 3KB          | 2005-03-26        |
| ChangeLog-2.6.11.7     | 6KB          | 2005-04-08        |
| ChangeLog-2.6.11.8     | 6KB          | 2005-04-30        |
| ChangeLog-2.6.11.9     | 3KB          | 2005-05-12        |
| ChangeLog-2.6.11.10    | 4KB          | 2005-05-17        |
| ChangeLog-2.6.11.11    | 15KB         | 2005-05-27        |
| ChangeLog-2.6.11.12    | 1KB          | 2005-06-12        |
| ChangeLog-2.6.12       | 1,024KB      | 2005-06-18        |



```
ChangeLog-2.6.7 + (WWW:verifier,kaist...hangeLogWchangeloadescriptions) = 65111
파일(F) 편집(E) 도구(T) 문법(S) 버퍼(B) 창(W) 도움말(H)
18349 <akpm@osdl.org>
18350 [PATCH] fix possible NULL pointer in fs/ext3/super.c.
18351
18352 From: "Luiz Fernando N. Capitulino" <lcapitulino@prefeitura.sp.gov.br>
18353
18354 In fs/ext3/super.c::ext3_get_journal() at line 1675 'journal' can be NULL,
18355 but it is not handled right (detect by Coverity's checker).
18356
18357 Signed-off by: Luiz Capitulino <lcapitulino@prefeitura.sp.gov.br>
18358 Signed-off-by: Andrew Morton <akpm@osdl.org>
18359 Signed-off-by: Linus Torvalds <torvalds@osdl.org>
18360
18361 <akpm@osdl.org>
18362 [PATCH] sched: balance-on-exec fix
18363
18364 From: Jack Steiner <steiner@sgi.com>
18365
18366 It looks like the call to sched_balance_exec() from do_execve() is in the
18367 wrong spot. The code calls sched_balance_exec() before determining whether
18368 "filename" actually exists.
18369
18370 In many cases, users have several entries in $PATH. If a full path name is
18371 not specified on the "exec" call, the library code iterates thru the files
18372 in the PATH list until it finds the program. This can result in numerous
18373 migrations of the parent process before the program is actually found.
18374
18375 Signed-off-by: Ingo Molnar <mingo@elte.hu>
18376 Signed-off-by: Andrew Morton <akpm@osdl.org>
18377 Signed-off-by: Linus Torvalds <torvalds@osdl.org>
```

## To Do List (2/2)

- You have to show traceability between your implementation and your design
  - Each state diagram can be used as a test oracle for unit/component tests
  - Similarly, activity diagram and sequence diagram can be used for component/integration/acceptance testing
- Do **not** forget to submit the revised design document

Ex. Security function

