Addressing Non-Functional Requirements in Mobile Applications

CSE 5236: Mobile Application Development
Instructor: Adam C. Champion, Ph.D.
Course Coordinator: Dr. Rajiv Ramnath
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Non-Functional Requirements (NFRs)

- AKA quality/design requirements
  - Building the app right (as opposed to the “right app” from functional requirements)
- Typical NFRs:
  - Performance
  - Availability
  - Scalability
  - Usability
  - Security
  - Modifiability
  - Maintainability and testability
  - Cost
- (Almost) always involves tradeoffs
  - Some combinations are aligned (e.g. security and availability)
Key NFRs for Mobile Devices

• Performance
• Responsiveness (not the same as performance)
• Energy (not covered here, see PowerManager class
  http://developer.android.com/reference/android/os/PowerManager.html and
• Security
Systematic Steps Towards Meeting NFRs

• “Quantify” for the app
• Make appropriate architectural decisions: often pre-determined by the underlying architecture of the implementation framework (e.g., the Android SDK)
• Optimize tactically based on real measurements
Architectural Decisions in Tic-Tac-Toe

• Java/Kotlin – reduced cost of development
• Data storage tactics:
  – Preferences: cost of development
  – SQLite: Reliability, queries faster than inserts suited for login use case.
• Data transfer in JSON
• 2-D graphics for speed
Tactical Optimizations Used in Tic-Tac-Toe

• Used variables to cache data retrieved from collections (e.g. arrays)
• Avoided internal use of getters and setters
• Reduced heap access: avoid creating unnecessary objects (see use of Singleton for X, O and Blank symbols)
• Used static final for constants (allows inlining of constants)
• Leveraged optimizations in framework libraries
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Optimize Performance with Profiler (1)

1. Connect Android device to dev machine
2. Click “Android Profiler” icon;
3. App starts running on device
4. Profile CPU use, memory use, etc.
Optimize Performance with Profiler (2)

1. Generate a method trace by pressing Record button.
2. Use the app “as normal”.

![Profiler GUI](image)
Method Trace View

- Method trace window appears.
- Find slow parts of program and investigate...
onDraw(), getBitmapForSymbol()

App is using 11.71% of CPU 😞

These methods are using CPU heavily here…
Looking Closely: onDraw()

// Board.java
public void onDraw() {
    ...
    for (int i = 0; i < GameGrid.SIZE; i++) {
        for (int j = 0; j < GameGrid.SIZE; j++) {
            Bitmap symSelected =
                getBitmapForSymbol(grid.getValueAtLocation(i, j));
            offsetX = (int)(((width - symSelected.getWidth())/2) + (i * width));
            offsetY = (int)(((height - symSelected.getHeight())/2) + (j * height));
            canvas.drawBitmap(symSelected, offsetX, offsetY, ditherPaint);
        }
    }
    ...
}

// Only considering Java here. Kotlin optimization is similar.
Looking Closely:
getBitmapForSymbol()

// Board.java
...

public Bitmap getBitmapForSymbol(Symbol aSymbol) {
    try {
        Resources res = getResources();
        sSymX = BitmapFactory.decodeResource(res, R.drawable.x);
        sSymO = BitmapFactory.decodeResource(res, R.drawable.o);
        sSymBlank = BitmapFactory.decodeResource(res, R.drawable.blank);
    } catch (OutOfMemoryError ome) {

        Bitmap symSelected = sSymBlank;

        if (aSymbol == Symbol.SymbolXCreate())
            symSelected = sSymX;
        else if (aSymbol == Symbol.SymbolOCreate())
            symSelected = sSymO;
        return symSelected;
    }
}
Optimizing `getBitmapForSymbol()`

```java
static Bitmap symX = null, symO = null, symBlank = null;
static boolean sDrawablesInitialized = false;

public Bitmap getBitmapForSymbol(Symbol aSymbol){
    if (!sDrawablesInitialized) {
        Resources res = getResources();
        symX = BitmapFactory.decodeResource(res, R.drawable.x);
        symO = BitmapFactory.decodeResource(res, R.drawable.o);
        symBlank = BitmapFactory.decodeResource(res, R.drawable.blank);
        sDrawablesInitialized = true;
    }
    Bitmap symSelected = symBlank;
    if (aSymbol == Symbol.SymbolXCreate())
        symSelected = symX;
    else if (aSymbol == Symbol.SymbolOCreate())
        symSelected = symO;
    return symSelected;
}
```
After Optimization
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Maximize Battery Life

• Reducing computation (same techniques as for performance)
• Reducing network usage
  – Minimizing data services
  – Minimizing location services
• Managing display brightness
Minimize Network Use: Java

• Check for network availability

```java
private boolean hasNetworkConnection() {
    ConnectivityManager connectivityManager =
        (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo =
        connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
    boolean isConnected = true;
    boolean isWifiAvailable = networkInfo.isAvailable();
    boolean isWifiConnected = networkInfo.isConnected();
    networkInfo =
        connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
    boolean isMobileAvailable = networkInfo.isAvailable();
    boolean isMobileConnected = networkInfo.isConnected();
    isConnected = (isMobileAvailable && isMobileConnected) ||
        (isWifiAvailable && isWifiConnected);
    return isConnected;
}
```

• Use compact data formats (JSON)
Minimize Network Use: Kotlin

• Checking for network availability:

```kotlin
private fun hasNetworkConnection(): Boolean {
    val connectivityManager =
        activity.applicationContext.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    var networkInfo = connectivityManager
        .getNetworkInfo(ConnectivityManager.TYPE_WIFI)
    var isConnected = true
    val isWifiAvailable = networkInfo.isAvailable
    val isWifiConnected = networkInfo.isConnected
    networkInfo = connectivityManager
        .getNetworkInfo(ConnectivityManager.TYPE_MOBILE)
    val isMobileAvailable = networkInfo.isAvailable
    val isMobileConnected = networkInfo.isConnected
    isConnected = (isMobileAvailable && isMobileConnected) || (isWifiAvailable && isWifiConnected)
    return isConnected
}
```
Minimize Location Services: Use Last Known Location

```java
public Location getBestCurrentLocation(){
    Location myLocation=null;
    myLocation = manager.getLastKnownLocation(bestProvider);
    if (myLocation == null) {
        myLocation = manager.getLastKnownLocation("network");
    }
    if (myLocation != null) {
        System.out.println("GeoLocation is>"+
            myLocation.toString()+"<");
        thisLocation = myLocation;
    }
    return thisLocation;
}
```
Minimize Location Services: Use Cheapest Provider

class GeoLocation
{
    public GeoLocation(Context theContext)
    {
        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_COARSE);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        LocationManager manager =
            (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        String cheapestProvider = myLocationManager.getBestProvider(c, true);
        registerForLocationUpdates();
    }
}
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Responsiveness: Threading: Ex. (1)

```java
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork();
            // user written method
            // do something with the image ...
        }
    }).start();
}
```
public void onCreate(Bundle savedInstanceState) {
    ...
    // Launch a new thread for the splash screen
    Thread splashThread = new Thread() {
        @Override public void run() {
            try {
                int elapsedTime = 0;
                while(active && (elapsedTime < splashTime)) {
                    sleep(sleepTime);
                    if(active) elapsedTime = elapsedTime + timeIncrement;
                }
            } catch(InterruptedException e) {
                // do nothing
            } finally {
                finish();
                startActivity(new Intent("com.wiley.fordummies.androidsdk.tictactoe.Login"));
            }
        }
    }
    splashThread.start();
}
Threading: Ex. (2): SplashScreen (2)

```java
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        active = false;
    }
    return true;
}
```
private void scheduleAndroidsTurn() {
    System.out.println("Thread ID in scheduleAndroidsTurn:" +
                      Thread.currentThread().getId());
    board.disableInput(); // prevents user input while machine is thinking
    if(!testMode){
        Random randomNumber = new Random();
        Handler handler = new Handler();
        handler.postDelayed( //Posts callback with a random delay.
            new Runnable()
                {
                    public void run()
                        {
                            androidTakesATurn();
                        }
                }, ANDROIDTIMEOUTBASE + randomNumber.nextInt(ANDROIDTIMEOUTSEED));
    } else {
        androidTakesATurn();
    }
}
public class HelpWithWebView extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedInstanceState) {
        String URL=null;
        super.onCreate(savedInstanceState);
        setContentView(R.layout.helpwithwebview);
        WebView helpInWebView=null;
        helpInWebView = (WebView) findViewById(R.id.helpwithwebview);
        View buttonExit = findViewById(R.id.button_exit);
        buttonExit.setOnClickListener(this);
        Bundle extras = getIntent().getExtras();
        if(extras !=null){URL = extras.getString("URL"); } 
        helpInWebView.loadUrl(URL); // runs in a separate thread
    }
    ...
}
The Android Thread Model

- Main thread usually the UI thread (but sometimes not – see below)
- SDK NOT thread-safe
  - Other threads should NOT manipulate UI
  - Only do computation, then offer result to UI thread
- API to access UI thread:
  - `Activity.runOnUiThread(Runnable myRunnable)` runs the specified runnable object on the UI thread. (See `GameSessionTest.java`)
  - `View.post(Runnable myRunnable)` causes the Runnable to be added to the message queue to be run by the UI thread as it processes all its messages.
  - `View.postDelayed(Runnable, long)` adds Runnable to message queue after specified period.
  - The `Handler` class lets you perform the preceding `post(...) and postDelayed(...)` operations when you don’t have access to an active View. (see `GameSession.java`)
- UI thread subordinated to unit test thread (see section on `Testing`)
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Security Considerations for Mobile Devices

- Devices store valuable personal information
- Larger security “footprint” – more attack surfaces ⟹ more vulnerabilities
  - Existing threats are magnified (e.g. poorly secured browsers and mobile web sites)
  - Installed apps are sources of insecurity.
    - More apps. Unknown trustworthiness of authors in an open marketplace.
  - Sharing between apps.
  - Private data left behind on file system
- Device is inherently less secure
  - Portable
  - No user login
  - Typically weaker passwords used (due to difficulty of data entry)
  - Less readable screen and environmental distractions means security indications could be ignored
- Lesson: App developers have to share responsibility for security
Systematic Steps to App Security

- **Do not**: randomly implement security “stuff”
- Instead, brainstorm and define a threat model
  - What are the assets (data)? What is their value?
  - What are the attacks (theft, denial of service)? Where can the attacks originate (network, other apps, browser)
- Identify security tactics:
  - Detection: Determining that an attack is in progress, or loss has taken place.
  - Resistance: Making the loss more difficult to occur.
  - Mitigation: Limiting the degree of loss or breach.
  - Recovery: Restoration from loss.
- Implement tactics using security techniques:
  - Authentication (e.g. two-factor, certificates)
  - Access control (e.g. file ownership, encryption, certificates)
  - Audit trail (e.g. logs)
  - Data integrity (e.g. checksums, encryption)
  - Non-repudiation (e.g. logs, certificates)
Android-Specific Security Considerations

• Good: “Privilege-supported” OS
  – Processes “sandboxed” in user space
  – User files and databases are removed on uninstallation
  – Apps must request and be granted permissions (on installation): to system resources, content providers, resources of other apps
  – Apps must be “signed” by developer (however, self-signing allowed!)
  – Google verifies new apps installed in Android 4.3+

• Bad:
  – No security through obscurity: Linux is open-source, APK file can be freely inspected, Java can be decompiled
  – Limited vetting process on Google Play (tests apps via QEMU full-system emulator*)
  – Privileges are enforced by installer (runtimes of hacked phones may not enforce privileges)

• Things to look out for:
  – Leaving private data in files on the device and its SD card
  – Database hacking techniques – SQL injection
  – Your app being the Trojan horse
  – Secret literals left in code (e.g. special passwords)
  – Use of reversible security algorithms

Examples of Permission Requests

- `<uses-permission
    android:name="android.permission.READ_CONTACTS"/>
- `<uses-permission android:name="android.permission.INTERNET"/>
- `<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
- `<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>
- `<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
- `<uses-permission android:name=
    "com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY"/>
    – Example of a custom permission

Note: Permission elements must be *outside* the `<application>` block and *inside* the `<manifest>` block of the AndroidManifest.xml
Custom Permissions – Definition and Placement

Permission must be declared:

```xml
<permission
    android:name="com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY"
    android:label="Launch Tic-Tac-Toe Activity"
    android:description="@string/permission_launch_activity"
    android:protectionLevel="normal"
/>
```

Then placed:

In the AndroidManifest.xml file outside the <application> block and inside the <manifest> block (same as <uses-permission> elements).
Custom Permission – Declaration of Need

• Declare need using `android:permission` attribute in activity definition in manifest file:

```xml
<activity
    android:name=".Login"
    android:label="@string/app_name"
    android:launchMode="standard"
    android:screenOrientation="portrait"
    android:permission="com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY">
    ...
</activity>
```
Custom Permission – Request

- `<uses-permission
  android:name="com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY"/>

- Requested in any separate package

- But also in containing package!
Permission Checking in Android

- When a call is made to a system function: To prevent an unauthorized invocation
- When starting an Activity: To prevent an unauthorized application from launching the Activity of other applications
- When sending or receiving Broadcasts: To determine who can receive a Broadcast or send it to you
- When accessing, and operating on, a Content Provider: To prevent an unauthorized app from accessing the data in the Content Provider
- When binding to, or starting, a Service: To prevent an unauthorized application from using the Service.
Example logcat Entries During Permission Failures

02-28 12:48:00.864: ERROR/AndroidRuntime(378):
    java.lang.SecurityException: Permission Denial: starting Intent {
        act=com.wiley.fordummies.androidsdk.tictactoe.Login
        cmp=com.wiley.fordummies.androidsdk.tictactoe/.Login } from ProcessRecord{407740c0
        378:com.wiley.fordummies.androidsdk.tictactoe/10033} (pid=378, uid=10033)
    requires
        com.wiley.fordummies.androidsdk.tictactoe.permission.LAUNCHACTIVITY

02-28 21:04:39.758: ERROR/AndroidRuntime(914): at
    com.wiley.fordummies.androidsdk.tictactoe.SplashScreen$1.run
    (SplashScreen.java:36)
Example logcat Entries for Permission Definition or Placement Errors

02-28 16:53:09.838: DEBUG/PackageManager(77): Permissions:
    com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY

02-28 17:04:18.888: WARN/PackageParser(77): Unknown element under <application>:

    permission at /data/app/vmdl1654102309.tmp Binary XML file line #11
02-28 17:04:20.438: WARN/PackageManager(77): Unknown permission
    com.wiley.fordummies.androidsdk.tictactoe.LAUNCHACTIVITY in package
    com.wiley.fordummies.androidsdk.tictactoe
SQLite Security – SQL Injection

• Entry field: Name: <Enter Name>
• Intended query:
  – `SELECT e-mail FROM user_information WHERE NAME='Bob'`
• Attacker enters string:
  – ‘Bob’; `SELECT table_names FROM user_tables`
• Query becomes:
  – `SELECT e-mail FROM user_information WHERE name='Bob'; SELECT table_names FROM user_tables`
• Attacker knows all the tables. Augh!
private static final String TABLE_NAME = "Accounts";
...
private static final String INSERT = "insert into " +
    TABLE_NAME + "(name, password) values (?, ?)" ;
...
public DatabaseHelper(Context context) {
...
    this.insertStmt = this.db.compileStatement(INSERT);
...
}
...
public long insert(String name, String password) {
    this.insertStmt.bindString(1, name);
    this.insertStmt.bindString(2, password);
    return this.insertStmt.executeInsert();
}
General Rule – Minimize Vulnerable Spots in App

• Don’t hardwire “secrets” in code.
• Mask sensitive data entry (e.g. passwords)
• Encrypt files
• Don’t write unnecessary temporary files
• Use bind variables
• Ask for the least permissions
• Create checkpoints of application data
• Log data (of course, encrypt your logs)
• Keep Intent Filters specific so Activities don’t respond to generic Intents
• Don’t rely only on permissions when sensitive data is accessed – prompt user.
Outline

• Non-Functional Requirements
• Optimize Performance with Profiler
• Maximize Battery Life
• Optimize for Responsiveness
• Improve App Security
• Testing
Creating Unit Tests (1)

• In Android Studio, right-click the project name, select app, click on Dependencies tab, click “+” icon, select “Library Dependency”, then type “junit” into the dialog

• Create test classes under <project-name>/app/src/androidTest/java/<package-name>

• Set up test run configuration (of type Android Test)
Creating Unit Tests (2)
Creating Unit Tests (3)
Passed Unit Test
Failed Unit Test

```java
public void testUIThreadTest() {
    System.out.println("Thread ID in testUI:" + Thread.currentThread().getId());
    board.requestFocus();
    MotionEvent newMotionEvent = MotionEvent.obtain((long)1,
        (long)1,
        MotionEvent.ACTION_DOWN,
        (float) 53.0,
        (float) 53.0,
        0);
    board.dispatchTouchEvent(newMotionEvent);
    assertEquals(gameSessionActivity.getPlayCount(), 2);
}
```
Unit Test Class

```java
public class GameSessionTest extends ActivityInstrumentationTestCase2 {
    // Template Class
    private GameSession gameSessionActivity; // Activity to be tested
    private Board board; // Member variable of activity
    private Instrumentation gameSessionActivityInstrumentation = null;
    // Data for the tests - touch coordinates
    final float x[]={(float)56.0, (float) 143.0, (float) 227.0};
    final float y[]={(float)56.0, (float) 143.0, (float) 227.0};
    int i = 0;
    public GameSessionTest() {...} // Constructor
    protected void setUp() throws Exception {...} // Setup - gets member variables
    public void testPreconditions() {...} // Test 1
    public void testUI() {...} // Test 2
    @UiThreadTest // Annotation to force the test to run in the UI thread
    public void testUIThreadTest(){...} // Test 3
    protected void tearDown() throws Exception {...} // Cleanup
}
```
Constructor

```java
public GameSessionTest() {
    super("com.wiley.fordummies.androidsdk.tictactoe.GameSession",
            GameSession.class);
}
```
protected void setUp() throws Exception {
    // access to member variables
    super.setUp();
    setActivityInitialTouchMode(false);
    gameSessionActivityInstrumentation = getInstrumentation();
    gameSessionActivity = getActivity();
    board = (Board) gameSessionActivity.findViewById(R.id.board);
}
...
protected void tearDown() throws Exception {
    // cleans up
    gameSessionActivity.finish();
}
Test 1 – Test Preconditions

```java
public void testPreconditions() {
    assertNotNull(gameSessionActivity);
    assertNotNull(board);
}
```
Test 2 – Test User Interface

```java
public void testUI() {
    System.out.println("Thread ID in testUI:" + Thread.currentThread().getId());
    getActivity().runOnUiThread(new Runnable() { // Run on UI thread
        public void run() {
            System.out.println("Thread ID in TestUI.run:" + Thread.currentThread().getId());
            board.requestFocus();
            // Simulates touch event
            // Hint: Instrumented the onTouchEvent(MotionEvent event) to get good pixel values for touch. Why not call onTouchEvent of Board directly?
            MotionEvent newMotionEvent = MotionEvent.obtain((long)1, (long)1, MotionEvent.ACTION_DOWN, (float) 53.0, (float) 53.0, 0);
            board.dispatchTouchEvent(newMotionEvent); // Dispatches touch event
        }
    });
    assertEquals(gameSessionActivity.getPlayCount(), 2); // Assert 2 moves
}
```
Test 3 – Series of Moves

```java
final float x[] = {(float)56.0, (float)143.0, (float)227.0};
final float y[] = {(float)56.0, (float)143.0, (float)227.0};
int i = 0;
...
@UiThreadTest
public void testUIThreadTest() {
    System.out.println("Thread ID in testUI:" + Thread.currentThread().getId());
    board.requestFocus();
    for (i=0; i<3; i++){
        MotionEvent newMotionEvent = // Factory Method
            MotionEvent.obtain((long)1, (long)1,
                MotionEvent.ACTION_DOWN,
                (float) x[i], (float) y[i], 0);
        board.dispatchTouchEvent(newMotionEvent);
    }
    assertEquals(gameSessionActivity.getPlayCount(), 2);
}
Tests and Threading

- Must explicitly run certain tests on UI thread
  - Via Annotations
  - Via explicit command
- Main UI thread subordinated to unit test thread
- Main UI thread terminated when tests run
- Tasks queued for main UI thread may not launch!
private void scheduleAndroidsTurn() {
    System.out.println("Thread ID in scheduleAndroidsTurn:" +
    Thread.currentThread().getId());
    board.disableInput();
    if (!testMode) {
        Random randomNumber = new Random();
        Handler handler = new Handler();
        handler.postDelayed(new Runnable{
            public void run{
                androidTakesATurn();
            }
        },
        500 + randomNumber.nextInt(2000)
    } else {
        androidTakesATurn();
    }
}
Useful Links for Testing

• See: http://developer.android.com/reference/android/view/MotionEvent.html for details of MotionEvent class

• See: http://developer.android.com/reference/android/view/View.html for how to send an event to a View (Board is a subclass of View)

• http://blog.blundell-apps.com/android-gradle-app-with-robolectric-junit-tests/ (JUnit and Robolectric via Gradle and Android Studio)
References

• Chapter 8: Making Your Application Fast and Responsive, from *Android SDK 3 Programming for Dummies*


• Jon Bentley, *Writing Efficient Programs*, [www.crowl.org/lawrence/programming/Bentley82.html](http://www.crowl.org/lawrence/programming/Bentley82.html)


• Chapter 12: Effectively Using Your Integration Development Environment
  – Covers Eclipse features, logging and unit testing (in Eclipse)