Overview of Frameworks: Introduction

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II
Systems Programming for Android
• Summarize key framework concepts
Module Introduction

• Summarize key framework concepts
Module Introduction

- Summarize key framework concepts

Overview of Frameworks
Douglas C. Schmidt
Module Introduction

• Summarize key framework concepts
Module Introduction

- Summarize key framework concepts
Module Introduction

- Summarize key framework concepts
Overview of Frameworks

Module Introduction

- Summarize key framework concepts
- Give examples of frameworks related to Android
  - [developer.android.com](http://developer.android.com)
Overview of Frameworks: Part 1

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II
Systems Programming for Android
Learning Objectives of this Module

- Understand why hardware has historically improved more consistently than software
Learning Objectives of this Module

- Understand why hardware has historically improved more consistently than software
- Recognize key characteristics of frameworks that help improve software productivity & quality

We’ll give pithy examples of frameworks from Android to reify key concepts
Hardware == Better, Faster, Cheaper

- Processor & network performance has increased by many orders of magnitude in past decades

Single-core 10 Megahertz to 3+ Gigahertz multi-cores

1,200 bits/sec to 10+ Gigabits/sec
Overview of Frameworks

Hardware == Better, Faster, Cheaper

• Processor & network performance has increased by many orders of magnitude in past decades

Single-core 10 Megahertz to 3+ Gigahertz multi-cores

• Extrapolating these trends another decade or so yields high-performance commoditized hardware infrastructure
  • Processors with 100’s→1,000’s of cores
  • ~100 Gigabits/sec LANs
  • ~100 Megabits/sec wireless
  • ~10 Terabits/sec Internet backbone

www.dre.vanderbilt.edu/~schmidt/dedicate.html has more on commoditization
Overview of Frameworks

Douglas C. Schmidt

Software == Buggier, Slower, & More Expensive?

• Unfortunately, software quality & productivity hasn’t improved as rapidly or predictably as hardware.
Software == Buggier, Slower, & More Expensive?

- Unfortunately, software quality & productivity hasn’t improved as rapidly or predictably as hardware.
- This is particularly problematic for mission-critical concurrent & networked software-reliant systems.

See [www.dre.vanderbilt.edu/~schmidt/comm-lessons.html](http://www.dre.vanderbilt.edu/~schmidt/comm-lessons.html) for more info.
Why Hardware Improves Consistently

Advances in hardware & networks stem largely from maturation of *standardized & reusable* interfaces, protocols, & modeling tools.

- x86 chipsets
- TCP/IP switches
Why Software Fails to Improve as Consistently

In general, software has not been as standardized or reusable as hardware.

Historically software developers have manually rediscovered & reinvented “point solutions” that are expensive to develop, integrate, validate, & sustain.
Why Software Fails to Improve as Consistently

In general, software has not been as standardized or reusable as hardware.

Consequence: Small changes in software/hardware have a big (negative) impact on system quality & sustainability.
A framework is an integrated set of software components that collaborate to provide a reusable architecture for a family of related applications.

- Frameworks promote “systematic reuse” by factoring out many general-purpose & domain-specific services from traditional application responsibility.

www.dre.vanderbilt.edu/~schmidt/reuse-lessons.html has info on systematic reuse.
Software frameworks exhibit several key characteristics that differentiate them from other forms of systematic reuse.

www.dre.vanderbilt.edu/~schmidt/CACM-frameworks.html has more info
Key Characteristics of Frameworks

• They exhibit “inversion of control” via callbacks
• AKA, “Hollywood Principle”

See [www.dre.vanderbilt.edu/~schmidt/Coursera/articles/hollywood-principle.txt](http://www.dre.vanderbilt.edu/~schmidt/Coursera/articles/hollywood-principle.txt)
Key Characteristics of Frameworks

- They exhibit “inversion of control” via callbacks
- AKA, “Hollywood Principle”
Key Characteristics of Frameworks

- They exhibit “inversion of control” via callbacks
- They provide integrated domain-specific structures & functionality

Application-specific functionality
Key Characteristics of Frameworks

- They exhibit “inversion of control” via callbacks
- They provide integrated domain-specific structures & functionality
Key Characteristics of Frameworks

• They exhibit “inversion of control” via callbacks

• They provide integrated domain-specific structures & functionality
  - e.g., abstract & concrete classes, control flows, etc.

AsyncTask

- execute()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()

Params - Types used in background work

Progress - Types used when indicating progress

Result - Types of result

Hook methods
Template method
Key Characteristics of Frameworks

- They exhibit “inversion of control” via callbacks
- They provide integrated domain-specific structures & functionality
  - e.g., abstract & concrete classes, control flows, etc.
- They are “semi-complete” applications

Application-specific functionality

- Networking
- Database
- GUI
- Mobile Apps
- Social Media
- Electronic Trading
Key Characteristics of Frameworks

• They exhibit “inversion of control” via callbacks
• They provide integrated domain-specific structures & functionality
  • e.g., abstract & concrete classes, control flows, etc.
• They are “semi-complete” applications
Key Characteristics of Frameworks

- They exhibit “inversion of control” via callbacks
- They provide integrated domain-specific structures & functionality
  - e.g., abstract & concrete classes, control flows, etc.
- They are “semi-complete” applications
- Completing a framework involves instantiating objects & subclassing & overriding “hook methods”

```java
AsyncTask
execute()
onPreExecute()
doInBackground()
onProgressUpdate()
onPostExecute()
```

```java
DownloadAsyncTask
onPreExecute()
doInBackground()
onPostExecute()
```

Overridden hook methods
Summary

- The quality of software (and the productivity of software developers) has historically lagged hardware (and hardware developers)
Summary

• The quality of software (and the productivity of software developers) has historically lagged hardware (and hardware developers)

• Particularly for mission-critical concurrent & networked software
Summary

• The quality of software (and the productivity of software developers) has historically lagged hardware (and hardware developers)
• Particularly for mission-critical concurrent & networked software
• Much cost, effort, & defects stem from continuous rediscovery & reinvention of core concepts & components across software industry

$\text{\$\$\$\$\$\$\$\$\$\$\$}$
Summary

• The quality of software (and the productivity of software developers) has historically lagged hardware (and hardware developers)
  • Particularly for mission-critical concurrent & networked software
• Much cost, effort, & defects stem from continuous rediscovery & reinvention of core concepts & components across software industry
• Frameworks improve productivity & quality of software development by
  • Reifying proven software designs & implementations in selected domains
Summary

- The quality of software (and the productivity of software developers) has historically lagged hardware (and hardware developers)
- Particularly for mission-critical concurrent & networked software
- Much cost, effort, & defects stem from continuous rediscovery & reinvention of core concepts & components across software industry
- Frameworks improve productivity & quality of software development by
  - Reifying proven software designs & implementations in selected domains
  - Amortizing quality assurance efforts & artifacts

www.dre.vanderbilt.edu/scoreboard
Summary

• The quality of software (& the productivity of software developers) has historically lagged hardware (& hardware developers)
• Particularly for mission-critical concurrent & networked software
• Much cost, effort, & defects stem from continuous rediscovery & reinvention of core concepts & components across software industry
• Frameworks improve productivity & quality of software development by
  • Reifying proven software designs & implementations in selected domains
  • Amortizing quality assurance efforts & artifacts

We have a long way to go to match hardware engineers use of modeling tools

w3.isis.vanderbilt.edu/projects/gme
Overview of Frameworks: Part 2

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives of this Module

- Understand how frameworks compare with other systematic reuse techniques
Overview of Frameworks

Learning Objectives of this Module

- Understand how frameworks compare with other systematic reuse techniques
- Recognize the different categories of frameworks

Black-box
White-box

Again, we give pithy examples of frameworks from Android to reify key points
Comparing Systematic Reuse Techniques

- **Class Library Architecture**
  - Class is a reusable implementation unit in an OO language
  - Classes are typically passive

See [en.wikipedia.org/wiki/Library_(computing)#Object_and_class_libraries](en.wikipedia.org/wiki/Library_(computing)#Object_and_class_libraries)
Overview of Frameworks

Douglas C. Schmidt

Comparing Systematic Reuse Techniques

• **Class Library Architecture**
  - Class is a reusable implementation unit in an OO language
  - Classes are typically passive

• **Framework Architecture**
  - Framework is integrated set of classes that collaborate to form a reusable architecture for a family of apps
  - Frameworks own the event loop(s)

See st-www.cs.illinois.edu/users/johnson/frameworks.html
Comparing Systematic Reuse Techniques

- **Class Library Architecture**
  - Class is a reusable implementation unit in an OO language
  - Classes are typically passive

- **Framework Architecture**
  - Framework is an integrated set of classes that collaborate to form a reusable architecture for a family of apps
  - Frameworks reify pattern languages

- **Component-based & Service-Oriented Architecture**
  - Component is an encapsulation unit with one or more interfaces that provide clients with access to services
  - Components can be deployed & configured via meta-data contained in assemblies

See [www.dre.vanderbilt.edu/~schmidt/report-doc.html](http://www.dre.vanderbilt.edu/~schmidt/report-doc.html) for more info
Comparing Systematic Reuse Techniques

- **Class Library Architecture**
  - Class is a reusable implementation unit in an OO language
  - Classes are typically passive

- **Framework Architecture**
  - Framework is an integrated set of classes that collaborate to form a reusable architecture for a family of apps
  - Frameworks reify pattern languages

- **Component-based & Service-Oriented Architecture**
  - Component is an encapsulation unit with one or more interfaces that provide clients with access to services
  - Components can be deployed & configured via meta-data contained in assemblies

Frameworks are generally more flexible/powerful than other systematic reuse techniques, but also more complicated to develop & use.
Categories of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects.
- Framework elements typically reused by parameterizing & assembling objects.
**Categories of Frameworks**

- **Black-box frameworks** only require understanding external interfaces of objects.
- Framework elements typically reused by parameterizing & assembling objects.

- **White-box frameworks** require understanding the framework implementation to some degree.
- Framework elements typically reused by subclassing & overriding.

[www.laputan.org/drc/drc.html](http://www.laputan.org/drc/drc.html) has more on black-box & white-box frameworks.
Overview of Frameworks

Categories of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects
- **White-box frameworks** require understanding the framework implementation to some degree
  - Framework elements typically reused by subclassing & overriding

- Each category of OO framework uses different sets of patterns

- Black-box frameworks reply heavily on object composition patterns, such as *Strategy* & *Decorator*

---

en.wikipedia.org/wiki/Strategy_pattern has more on the *Strategy* pattern
Categories of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects
- **White-box frameworks** require understanding the framework implementation to some degree
  - Framework elements typically reused by subclassing & overriding

- Each category of OO framework uses different sets of patterns

- White-box frameworks rely heavily on inheritance-based extensibility patterns, such as *Template Method & State*

en.wikipedia.org/wiki/Template_method has more on *Template Method* pattern
Categories of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects

- **White-box frameworks** require understanding the framework implementation to some degree
  - Framework elements typically reused by subclassing & overriding

- Each category of OO framework uses different sets of patterns
- Many frameworks fall in between white-box & black-box categories
Categories of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects

- **White-box frameworks** require understanding the framework implementation to some degree
  - Framework elements typically reused by subclassing & overriding

- Each category of OO framework uses different sets of patterns
- Many frameworks fall in between white-box & black-box categories

In general
- White-box frameworks are easier to develop, but harder to use
Categories of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects
- **White-box frameworks** require understanding the framework implementation to some degree
  - Framework elements typically reused by subclassing & overriding
- Each category of OO framework uses different sets of patterns
- Many frameworks fall in between white-box & black-box categories

In general
- White-box frameworks are easier to develop, but harder to use
- Black-box frameworks are harder to develop, but easier to use
White-box Framework: Android AsyncTask

- Android’s AsyncTask provides a simple white-box framework to create long-running operations that need to communicate with the UI thread.

`developer.android.com/reference/android/os/AsyncTask.html` has AsyncTask info.
White-box Framework: Android AsyncTask

- Android’s AsyncTask provides a simple white-box framework to create long-running operations that need to communicate with the UI thread.
- Must be subclassed.

**Template method**

**Hook methods**

**AsyncTask**
- `execute`
- `onPreExecute`
- `doInBackground`
- `onProgressUpdate`
- `onPostExecute`

**DownloadAsyncTask**
- `onPreExecute`
- `doInBackground`
- `onPostExecute`

**Params** - Types used in background work
**Progress** - Types used when indicating progress
**Result** - Types of result
Class DownloadAsyncTask extends AsyncTask<String, Integer, Bitmap> {

    protected void onPreExecute() {
        dialog.display(); // Perform on UI thread
    }

    protected Bitmap doInBackground(String... url) {
        return downloadImage(url[0]); // Download in background thread
    }

    protected void onPostExecute(Bitmap bitmap) {
        performPostDownloadOperations(bitmap);
        dialog.dismiss(); // Perform on UI thread
    }
}

White-box Framework: Android AsyncTask

- Android's AsyncTask provides a simple white-box framework to create long-running operations that need to communicate with the UI thread.
- Must be subclassed.
- Hook methods can be overridden.

Perform on UI thread
White-box Framework: Android AsyncTask

- Android’s AsyncTask provides a simple white-box framework to create long-running operations that need to communicate with the UI thread
- Must be subclassed
- Hook methods can be overridden
- Instance must be created on the UI thread & can only be executed once

```java
public class ThreadedDownloadActivity extends Activity {
    ...
    public void runAsyncTask(View view) {
        final String url = 
            urlEditText.getText().toString();

        new DownloadAsyncTask().execute(url);
    }
    ...
}
```

UI thread calls template method to trigger image download in a new AsyncTask
Black-box Framework: Android AsyncTask

- Android’s AsyncTask provides a simple black-box framework for controlling the # & behavior of thread(s) running in background.

```
execute
onPreExecute
onPostExecute
execute
doInBackground
```

Black-box Framework: Android AsyncTask

- Android’s AsyncTask provides a simple black-box framework for controlling the behavior of thread(s) running in background.
- Client can select the desired Executor.

Serialization:

- `SERIAL_EXECUTOR`, `THREAD_POOL_EXECUTOR`, or custom Executor

Execution Flow:

- `onPreExecute`
- `execute`
- `onPostExecute`

Executor Options:

- `setDefaultExecutor (AsyncTask.THREAD_POOL_EXECUTOR)`

Result:

- Allows multiple long-running tasks to run in parallel in multiple threads.
Black-box Framework: Android AsyncTask

- Android’s AsyncTask provides a simple black-box framework for controlling the # & behavior of thread(s) running in background
- Client can select the desired Executor
- Executor treated as a “black-box”
  - i.e., only requires understanding of external interfaces

---

www.dre.vanderbilt.edu/~schmidt/PDF/ICCDS.pdf has C++ black-box example
Summary

- Frameworks are powerful—but can be hard to develop & use by app developers due to inherent/accidental complexities of various domains.
Summary

- Frameworks are powerful—but can be hard to develop & use by app developers due to inherent/accidental complexities of various domains.
- Patterns (especially pattern languages) help to alleviate many framework complexities.

www.dre.vanderbilt.edu/~schmidt/PDF/ORB-patterns.pdf has more info
Summary

- Frameworks are powerful—but can be hard to develop & use by app developers due to inherent/accidental complexities of various domains
- Patterns (especially pattern languages) help to alleviate many framework complexities
- It’s often better to use & customize “off-the-shelf” frameworks than to develop frameworks in-house
Overview of Frameworks

Summary

• Frameworks are powerful—but can be hard to develop & use by app developers due to inherent/accidental complexities of various domains

• Patterns (especially pattern languages) help to alleviate many framework complexities

• It’s often better to use & customize “off-the-shelf” frameworks than to develop frameworks in-house

• Components & services are easier for app developers to use, but aren’t as powerful or flexible as frameworks
• Frameworks are powerful—but can be hard to develop & use by app developers due to inherent/accidental complexities of various domains

• Patterns (especially pattern languages) help to alleviate many framework complexities

• It’s often better to use & customize “off-the-shelf” frameworks than to develop frameworks in-house

• Components & services are easier for app developers to use, but aren’t as powerful or flexible as frameworks

• Successful software projects are therefore often best organized using the “funnel” model

www.dre.vanderbilt.edu/~schmidt/PDF/Queue-04.pdf has more on frameworks
Overview of Frameworks: Part 3

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II
Systems Programming for Android
Learning Objectives of this Module

• Present *Scope, Commonality, & Variability (SCV)* analysis as a method for developing & applying software product-lines & frameworks
Learning Objectives of this Module

- Present *Scope, Commonality, & Variability* (SCV) analysis as a method for developing & applying software product-lines & frameworks

- Illustrate the application of SCV to Android
Overview of Software Product-Lines

- A *software product line* (SPL) is a form of systematic software reuse
- An SPL a set of software-intensive systems
- These systems share a common, managed set of features satisfying the specific needs of a particular market segment or mission
- They are developed from a common set of core assets in a prescribed way
Overview of Software Product-Lines

- A software product line (SPL) is a form of systematic software reuse
- An SPL a set of software-intensive systems
- These systems share a common, managed set of features satisfying the specific needs of a particular market segment or mission
- They are developed from a common set of core assets in a prescribed way
- Frameworks can help define & improve core SPL assets by factoring out many reusable general-purpose & domain-specific services from application responsibility

See [www.sei.cmu.edu/productlines](http://www.sei.cmu.edu/productlines) for more info on software product-lines
Scope, Commonality, & Variability Analysis

- Key software product-line & framework structure & behavior can be captured systematically via *Scope, Commonality, & Variability* (SCV) analysis
Overview of Frameworks

Scope, Commonality, & Variability Analysis

- Key software product-line & framework structure & behavior can be captured systematically via *Scope, Commonality, & Variability* (SCV) analysis.

- This process can be applied to identify commonalities & variabilities in a domain.
Scope, Commonality, & Variability Analysis

- Key software product-line & framework structure & behavior can be captured systematically via *Scope, Commonality, & Variability* (SCV) analysis.

- This process can be applied to identify commonalities & variabilities in a domain.

- Often used to guide the development & application of software product-lines & frameworks.

---

**Application Frameworks**

- Bundled & Third-Party Apps
- System Libraries
- Virtual Machine Runtime
- Operating System Kernel

**Product Variants**

- Product Variant 1
- Product Variant 2
- Product Variant 3
- Product Variant 4
Overview of Frameworks

Scope, Commonality, & Variability Analysis

- Key software product-line & framework structure & behavior can be captured systematically via *Scope, Commonality, & Variability* (SCV) analysis.

- This process can be applied to identify commonalities & variabilities in a domain.

- General method
  - Identify common portions of a domain & define stable interfaces (fairly easy).

```java
AsyncTask
execute()
onPreExecute()
doInBackground()
onProgressUpdate()
onPostExecute()
```
Overview of Frameworks

Scope, Commonality, & Variability Analysis

- Key software product-line & framework structure & behavior can be captured systematically via *Scope, Commonality, & Variability* (SCV) analysis.

- This process can be applied to identify commonalities & variabilities in a domain.

- General method
  - Identify common portions of a domain & define stable interfaces (fairly easy).
  - Identify variable portions of a domain & define stable interfaces (harder).

- Examples:

  ```java
  AsyncTask
  execute()
onPreExecute()doInBackground()onProgressUpdate()onPostExecute()
  ```

  ```java
  DownloadAsyncTask
  onPreExecute()doInBackground()onPostExecute()
  ```

  **Params** - Types used in background work
  **Progress** - Types used when indicating progress
  **Result** - Types of result.
Overview of Frameworks

Scope, Commonality, & Variability Analysis

• Key software product-line & framework structure & behavior can be captured systematically via Scope, Commonality, & Variability (SCV) analysis

• This process can be applied to identify commonalities & variabilities in a domain

• General method
  
  • Identify common portions of a domain & define stable interfaces (fairly easy)
  
  • Identify variable portions of a domain & define stable interfaces (harder)

• Create different implementations of the variable portions as plug-ins

Params – Types used in background work
Progress – Types used when indicating progress
Result – Types of result
Overview of Frameworks

Scope, Commonality, & Variability Analysis

- Key software product-line & framework structure & behavior can be captured systematically via Scope, Commonality, & Variability (SCV) analysis.

- This process can be applied to identify commonalities & variabilities in a domain.

- General method:
  - Identify common portions of a domain & define stable interfaces (fairly easy).
  - Identify variable portions of a domain & define stable interfaces (harder).
  - Create different implementations of the variable portions as plug-ins.

SOLID (object-oriented design) en.wikipedia.org/wiki/SOLID_(object-oriented_design) has more info.
Applying SCV to Android

- **Scope** defines the domain & context of Android & its various frameworks & components
  
  - e.g., Resource-constrained mobile devices
    
    - e.g., limited power, memory, processors, network, & price points
Applying SCV to Android

- **Scope** defines the domain & context of Android & its various frameworks & components

- e.g.,
  - Resource-constrained mobile devices
    - e.g., limited power, memory, processors, network, & price points
  - Touch-based user interfaces
Applying SCV to Android

- **Scope** defines the domain & context of Android & its various frameworks & components
  - e.g.,
    - Resource-constrained mobile devices
      - e.g., limited power, memory, processors, network, & price points
    - Touch-based user interfaces
  - (Largely) open-source, vendor- & hardware-agnostic ecosystem
Overview of Frameworks

Applying SCV to Android

**Scope** defines the domain & context of Android & its various frameworks & components

- e.g.,
  - Resource-constrained mobile devices
    - e.g., limited power, memory, processors, network, & price points
  - Touch-based user interfaces
  - (Largely) open-source, vendor- & hardware-agnostic ecosystem
  - Focus on installed-base of Java app developers

See [developer.android.com](https://developer.android.com) for more info on Android
• **Commonalities** describe the attributes common across all instances of Android

• **Common framework components**

  • e.g., Activities, Services, Content Providers, & Broadcast Receivers
Applying SCV to Android

- **Commonalities** describe the attributes common across all instances of Android
  - **Common framework components**
    - e.g., Activities, Services, Content Providers, & Broadcast Receivers
  - **Common application frameworks**
    - e.g., Activity Manager, Package Manager, Telephony Manager, Location Manager, Notification Manager, etc.
Overview of Frameworks

Applying SCV to Android

- **Commonalities** describe the attributes common across all instances of Android
  - **Common framework components**
    - e.g., Activities, Services, Content Providers, & Broadcast Receivers
  - **Common application frameworks**
    - e.g., Activity Manager, Package Manager, Telephony Manager, Location Manager, Notification Manager, etc.
  - **Common infrastructure**
    - e.g., Intent framework, Binder, Webkit, Hardware Abstraction Layer, OS device driver frameworks etc.
Applying SCV to Android

- **Variabilities** describe the attributes unique to different instantiations of Android

- **Product-dependent components**
  - e.g., different “look & feel” variants of vendor-specific user interfaces, sensor & device properties, etc.
Overview of Frameworks

Applying SCV to Android

- **Variabilities** describe the attributes unique to different instantiations of Android
  
  - **Product-dependent components**
    
    e.g., different “look & feel” variants of vendor-specific user interfaces, sensor & device properties, etc.

  - **Product-dependent component assemblies**

    e.g., different bundled apps, CDMA vs. GSM & different hardware, OS, & network/bus configurations, etc.

**SCV can also be applied recursively for all the Android frameworks & layers**
Summary

- **Scope, Commonality, & Variability (SCV) analysis is an advanced systematic reuse technique**
- It helps developers alleviate problems associated with maintaining many versions of the same product that have large amounts of similar software created to satisfy new & diverse requirements
Summary

- **Scope, Commonality, & Variability (SCV) analysis** is an advanced systematic reuse technique.
  - It helps developers alleviate problems associated with maintaining many versions of the same product that have large amounts of similar software created to satisfy new & diverse requirements.

- The frameworks in Android form software product-lines that enable systematic software reuse across a wide range of apps & infrastructure platforms.