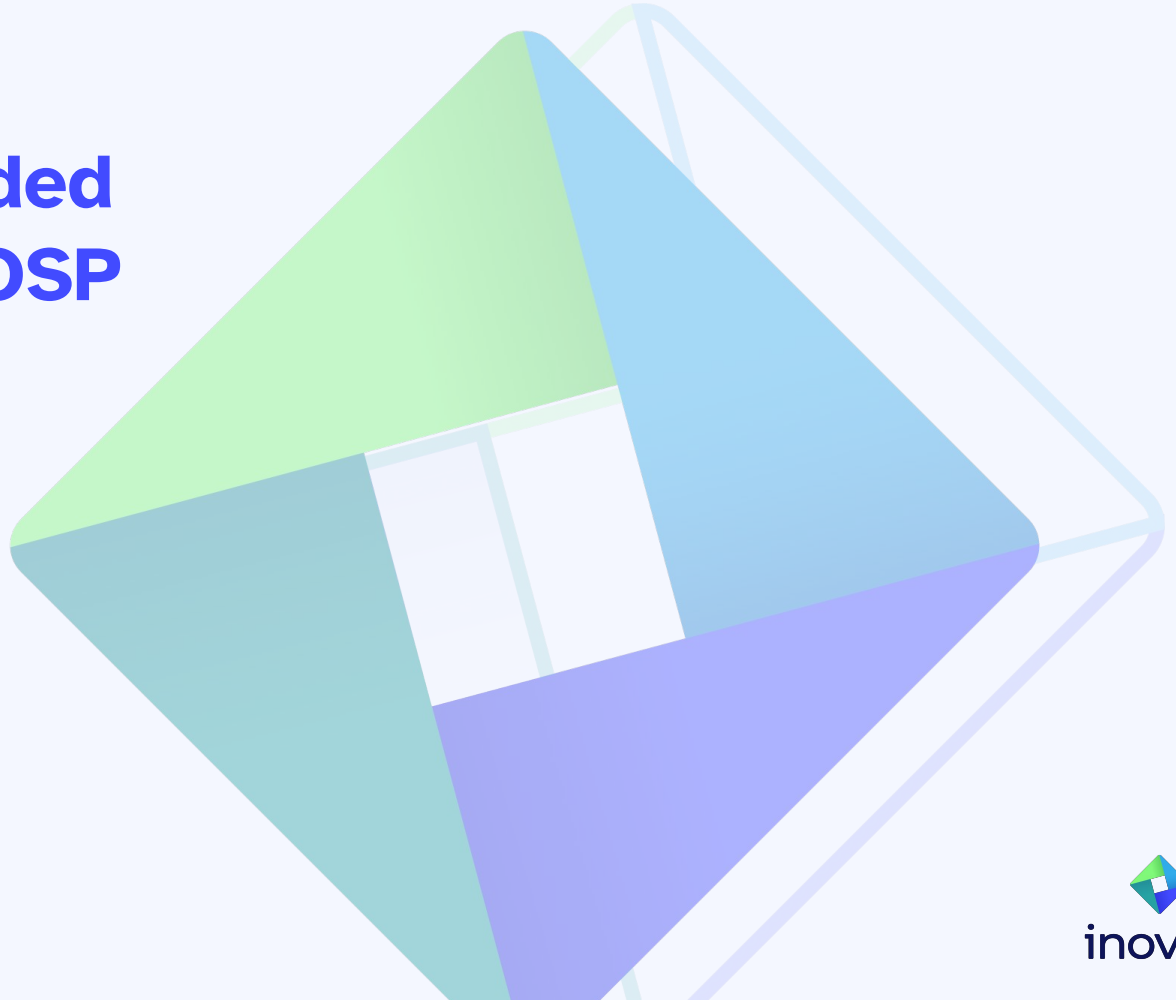


# Building Embedded Systems with AOSP

Why You Should Consider,  
Best Practices and Pitfalls

**Anna-Lena Marx**

*Meetup Erlangen, 14. Juni 2023*



inovex

# Anna-Lena Marx



## Embedded Systems Dev at inovex

- Linux Kernel
- Yocto
- Android Embedded



@Allegra3141



Allegra42

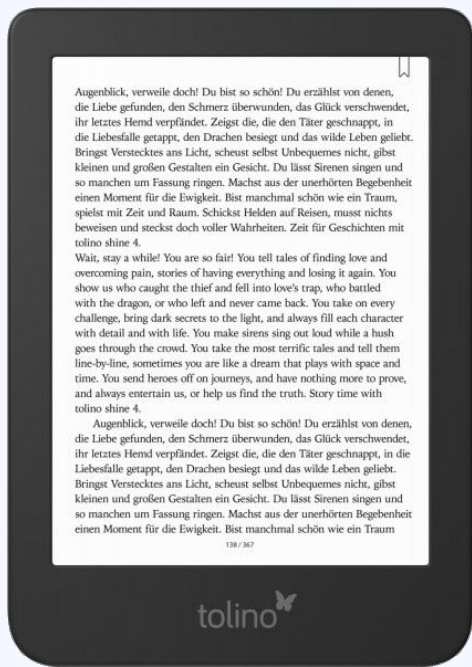


Allegra  
@social.linux.pizza

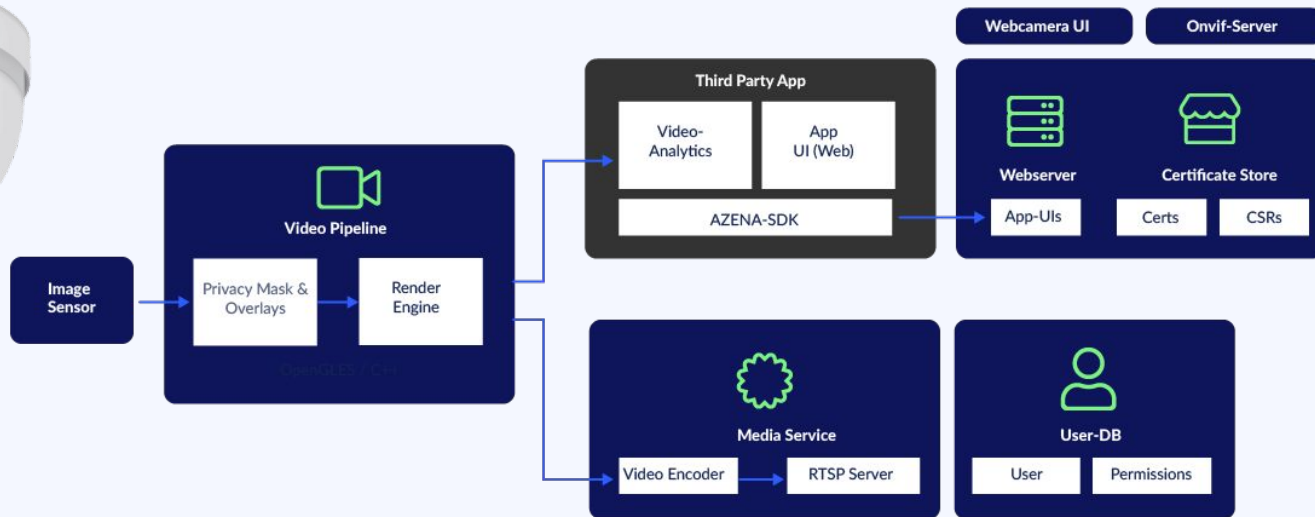
*B.Sc. Computer Science*

*M.Sc. Embedded Systems*

*B.Eng. Electrical Engineering - ongoing Hobby*



# azena



# Why you should consider AOSP for embedded systems

An introduction



inovex

# What's the Android Open Source Project?

Two levels of compatibility:

## **AOSP compatibility**

defined by Compatibility Definition Document (CDD)

## **Android compatibility**

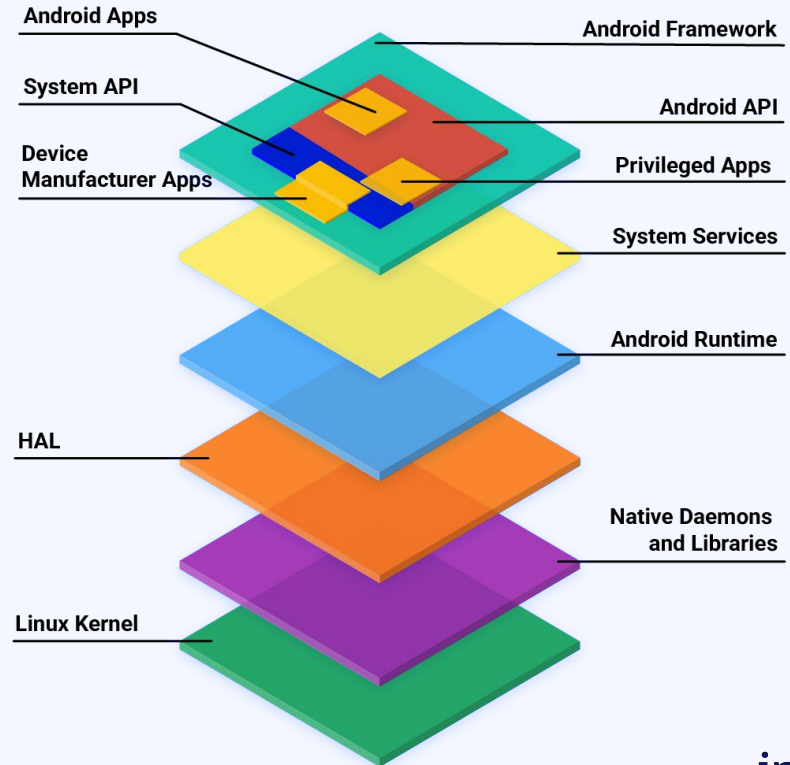
CDD plus

- Vendor Software Requirements (VSR)
- Vendor Test Suite (VTS)
- Compatibility Test Suite (CTS)

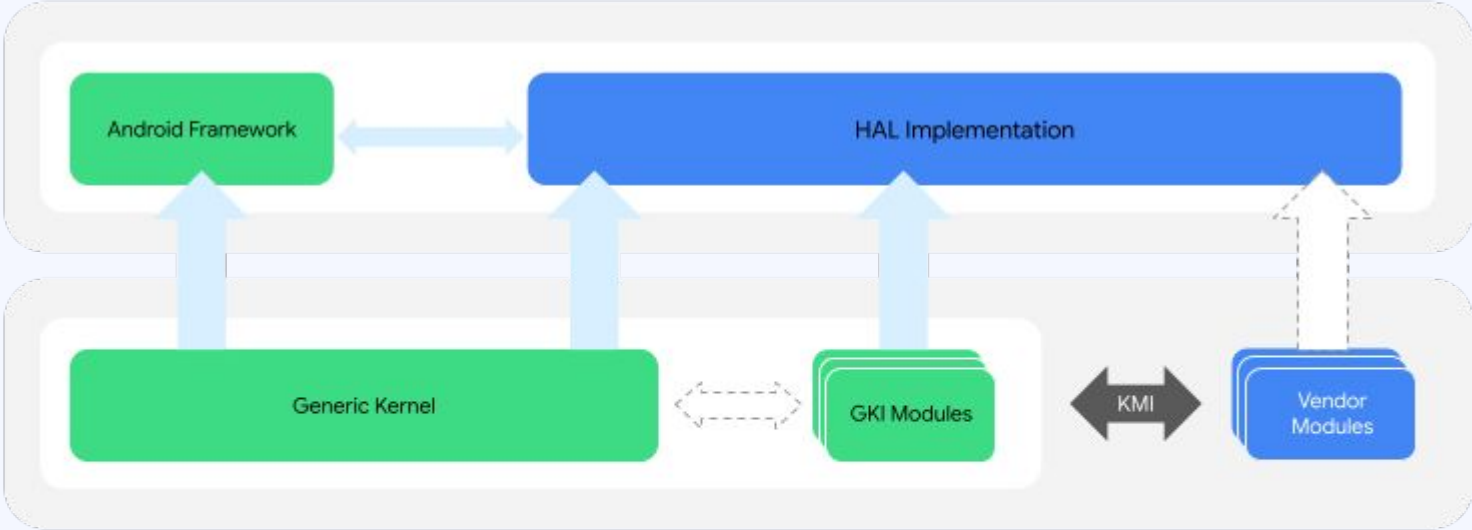
Additional step:

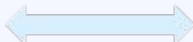

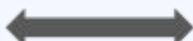
## **Licensing Google Mobile Services (GMS)**

- Google Apps (Youtube, Maps, Gmail, ...)



# Kernel philosophy



-  Stable Interfaces
-  Unstable Interfaces
-  Kernel Module Interface (KMI)  
(stable between Android Platform Releases)

 AOSP / Protected	 Vendor
---	--

# What the AOSP already does for you

The pro side





# What's included from a system engineers perspective

- provides a proven, stable and solid open source **platform**
- based on Linux
- graphics, video and media support
- connectivity
  - WiFi
  - Bluetooth
  - NFC, ...
- secure runtime environment for untrusted apps
- defined APIs between layers
- support for multiple target device types
- optimization for power-efficiency
- sophisticated Hardware Abstraction Layers (HAL)



# What's included from a security perspective

## On system level

- SELinux
- ASAN and sanitizer in general during build time
- dm-verity
- hardware keystore (Trustzone)
- OTA (over the air) update mechanism with A/B concept
- read-only system filesystem

## On app level

- sophisticated app concept with
  - secure runtime for untrusted apps
  - app isolation

# Benefits for app developers and users

- an UI/UX concept people are already familiar with
  - platform focuses on UX!
- well-known app developer ecosystem
  - standardized **APIs** with a good abstraction level
  - extensive set of system and third-party **libraries**
  - pretty good and in depth **documentation**
  - large eco system of existing apps
  - lots of *good* Android **app developers**



# The downsides of using AOSP



# Pain points of the AOSP itself

- very extensive code base
  - > 200 GB source code
  - needs potent build hardware and lots of compile time
- pretty high target system requirements
- high and increasing complexity
  - steep learning curve
  - lots of layers and abstractions within the system
- internals are continuously morphed
- build system chaos / changing build systems
- painful version updates

# Google, SoC vendors and OEMs

- AOSP is open-source but hard to contribute for people outside
- branding as Android and shipping Google apps require certification
- non-transparent partner ecosystem and license agreements
  
- SoC vendors and OEMs are not interested in selling low and medium quantities
  
- BSPs from SoC vendors
  - may need for a NDA to access the BSP
  - vendor specific, non-standard additions and build scripts
  - binary blobs (check licenses!)
  - quality depends on vendor

## General downsides

- focus on target device types
  - no support on other types or use-cases
  - clear focus on phones and UI-based devices (e.g. cars)
- intended for large volume products or own ecosystems
- OTAs needs own servers, management and good test runs
- few AOSP system engineers available

# When is using the AOSP a good choice?

And when not?





## Consider using the AOSP if...

- you want to build an **ecosystem**, that
  - targets various devices
  - runs on various hardware targets
  - is maybe open for third party app developers
- you need an app concept with strong isolation of untrusted apps
- you need the advantages of a sophisticated touchscreen UI
- you want to utilize the advantages of the media framework
- you need a GPL free system layer

# Don't use the AOSP

- if the overhead of AOSP does not give you clear advantages e.g.
  - you don't need a graphics/media stack at all
  - you don't care about the app concept
- if you're performance sensitive e.g.
  - boot time needs to be extremely short
- if you have a “small” use-case
  - with limited hardware resources
  - which is cost-sensitive
- if you don't have the person-power to maintain a full-blown AOSP system
  - you are strongly dependent to a SoC vendor (BSPs)
  - no open community support



# Best Practices, Recommendations and Pitfalls

Getting started

# Working with the source

- Use a code search engine like <https://cs.android.com/android>
- Make developing within AOSP smoother by importing it as Android Studio Project
  - `source build/envsetup.sh`
  - `Lunch <your-build-config>`
  - `make idegen && development/tools/idegen/idegen.sh`
  - Open android.ipr with Android Studio
    - Wait until indexing is finished
    - If there are issues, see <https://github.com/flutter/flutter-intellij/issues/1735#issuecomment-376918296>

<https://cs.android.com/android/platform/superproject/+/master:development/tools/idegen/README>

# Working with the source

## Working with the AOSP tree in Android Studio

- Allows general AS features (mostly for Java, but as well for native code)
  - Helps with type information
  - Finds usages
  - Nice for system development (Java services, native code, ...)
- Pretty comfortable for executing tests (gradle build files needed!)
- Allows (Java) debugging from IDE
  
- No support for building the whole AOSP from the IDE!



# Select a platform to start with

Phone, tablet, development board or emulator?

- Just want to start working with AOSP? Use a Google device!
- Emulator is limited in case of working with external hardware, kernel and drivers
- Dev boards are often outdated or not longer available (officially in AOSP supported ones)
- Use the target platform for your product if possible (Reference Board)
  - Be aware of availability issues
  - Differences between reference design and actual device platform
  - Vendor flavoured AOSP BSPs are ... different

# Vanilla AOSP vs. Board Vendor BSPs

## Vanilla AOSP

- Supported by Google
- Standard way of working (as documented)
- Latest versions available
- In general not suitable to build products
  - Google HW not available
  - Dev boards not suitable for production

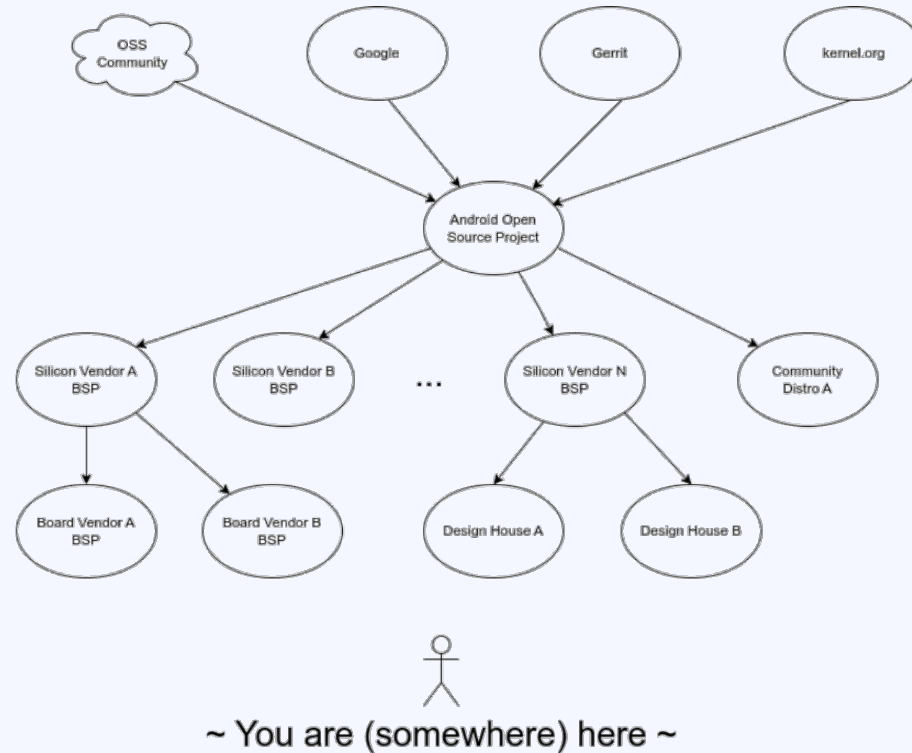
*Great to start*

## Board Vendor branded BSPs

- Supported by vendor
- Often with custom build scripts
- Diverging workflows (e.g. Qualcomm Kernel Build)
- Slow build due to custom logic
- Not all versions for all dev boards and platforms
- Updates are delayed
- NDAs, mostly always hard getting access

*Needed when building products*

# BSPs from h\*ll





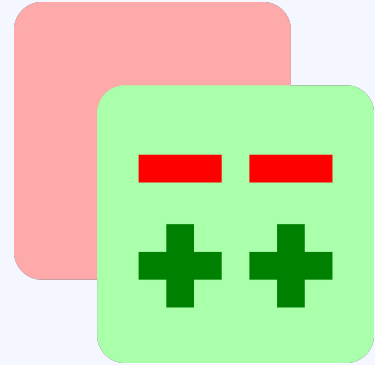
# Best Practices, Recommendations and Pitfalls

Tools and Workflow

# Version control and working in a multi-repo project

## Use **Gerrit Code Review!**

- proper handling of cross-repo commits
- enforces a clean way of working
  - e.g. with proper, meaningful commit messages
  - nice for code reviews
  - good overview of open change requests, open reviews, ...



**Gerrit** is a pretty ugly frontend in contrast to GitLab, GitHub, ...  
but the best tool when working in a multi-repo project as AOSP is!

# Continuous integration

## Use a **container setup!**

- can be reused within a CI pipeline
- reproducible between different devs and CI
- easier start for new developers



## Use a **CI pipeline!**

- Working in a multi-repo setup with different teams and changes in different repos have a large potential to break your environment
- forcing successful CI runs (including tests) as a requirement for merging reduce the risk!



# Continuous integration and automated tests

Your CI should include:

- building all your targets
- checks for code style (linting)
- testing
  - on real hardware and / or virtual devices
  - use / enhance existing test suites e.g.
    - Compatibility Test Suite (CTS)
    - Vendor Test Suite (VTS)
    - Trade Federation Testing Infrastructure (tradefed/TF)
  - write lots of tests!



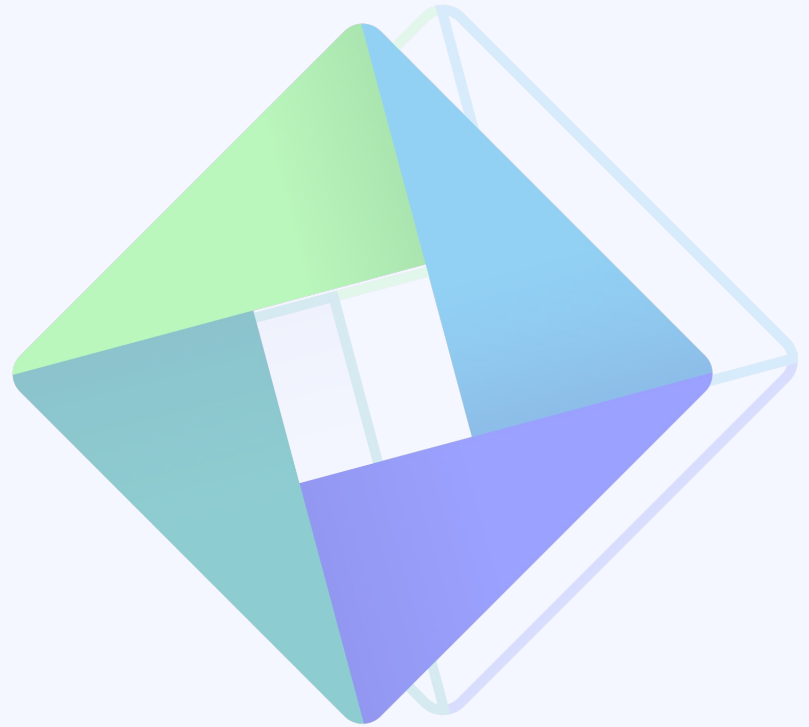
## Android Virtual Devices (AVD)

Tweak existing configuration to match your hardware device as good as possible!

- faster round-trips in development
- can be integrated with CI (e.g. being used for test runs)
- allow easier switching between versions, API levels, ...
- portable, home-office friendly

AVDs **do not** replace the need for testing and developing on real hardware, but they are a great convenience feature!

# Thank you!



**Anna-Lena Marx**

*Embedded Systems Dev*  
inovex GmbH  
Ludwig-Erhard-Allee 6  
76131 Karlsruhe  
[anna-lena.marx@inovex.de](mailto:anna-lena.marx@inovex.de)

## Further reading

- <https://www.inovex.de/de/blog/aosp-advanced-development-tricks/>
- <https://source.android.com/docs/core/architecture>
- <https://source.android.com/docs/core/architecture/kernel>
- <https://developer.android.com/studio/run/emulator>
- <https://www.gerritcodereview.com/index.html>
- <https://www.opersys.com/downloads/cc-slides/android-debug/slides-main-221201.html#/>
-