# Core Working Group Agenda 10/26/17 meeting.

Attended by Applications, Testing, Device Services, Core and DevOps WG leads. Additional attendees from IoTech, Dell, Cavium, Beechwood, Mainflux and others.

Discussion and action items as a result of meeting in RED.

## Old Business

This week, we are going to try to skip any status updates in order to get to some architectural discussion and items quickly. The purpose of today’s meeting is to have an open discussion about many of these architectural items, but then draw down to some conclusions before the end of today’s session. What we don’t complete will be carried over into the next meeting.

One item of Old Business to conclude before we begin

**Project Management Tool**

* Janko had the action item from last meeting to test GitHub Projects to see if tasks could be converted to issues and if that would resolve access issues.
* What would be the means to “tag” something an issue versus an actual bug
* Janko’s report to follow: Slides provided separately.

Jeremy – this seems workable. We don’t need the boards public – just the issues.

*Any conflict with GitHub proposed product changes?*

Jeremy – No, this shouldn’t mess up the process. Only makes board visible. Which is where we want the information anyway.

Tony – once you convert task to issue, its movable within board?

Janko – yes

Tony - Cool – works for me

Janko – once converted to issue, you can convert to file, technical solution integration, etc.

Andy – when you convert to issue, does it go against a specific repos?

Janko – drop down is available to apply it to a repos.

Andy – where would tasks for test be applied to? I.e. where would generic tasks go?

Janko – would have to have dummy repos or test repos.

**Jim Action Items – where to associate tasks? Label determination. What should they be? Work with Janko and Team to come up with strawman**

**Group unanimously approves and adopt’s Janko’s proposal to use GitHub Projects with Issues**

More action items may follow to set up tasks/boards/etc.

## New Business

Architectural Discussion on the following topics

**Shared libraries**

* *Yes or no*

Consensus is generally yes we want libraries

Tony – if we have the principal that allows to start from the ground up and not use libraries, that should be ok too.

If we find something in a library that doesn’t work, that’s bug and needs to be addressed as such.

Different language implementations will require by nature multiple “libraries” so as to prevent tight coupling.

We will need to figure out versioning on domain objects with regard to API changes.

Keith – Steve shared same opinions in email, but believes libraries are necessary.

Andy – to rip up core at the minute doesn’t make sense. New services may allow for not using shared library. Need to be focusing.

* *What constitutes tight coupling and how to avoid*

Tony – tight coupling is when a library everyone is using that forces everyone to update to that library – that is tight coupling.

*How about changes to core libraries when community makes decision to add something?*

Tony – if we add a field, shouldn’t cause an issue in JSON.

Brad – yeah you tend to ignore things you don’t understand. Additive/optional elements fine; backward compatibility is a must

Jeremy – direct calls to another API creates problem. Ex: calls to external service inside of a method that prevents swap out to another service.

Andy – example MongoDB swap out

Drasko – my only remark when rewriting in Go, data models that some models from Core were used in export services and this was confusing to me. Example: when rewriting Java in Go, then the Java class includes members that include some other domain (not core domain) because I didn’t want to touch core domain. Export domain -> core-domain required rewriting all these.

If core-domain already existed, would that have helped?

Drasko – we literally copied model in each micro service; this has drawbacks of course. Causing change everywhere when change in one service. This kind of implementation would not have caused this problem. How comfortable do we feel about importing core into export?

Export services are still part of EdgeX and the data models are still used to do serialization/deserialization. So I want to deserialize in local object from JSON. Where does this model come from?

*So if we decide not to use core library, we have to duplicate code and make sure that you, the non-library developer, keep up with core.*

Brad - Source of truth is core libraries

Tony – is it core models or APIs as in RAML

Drasko – agree with Tony spec is truth and code is implementation of that

Tony – could bound constants as part of JSON. Might be worthwhile to updating JSON version (3 to 7).

Action: We should be conscious in JSON version used.

Tony – should we create issues in GitHub for this?

Generally don’t want this intertwined with code/implementation bugs and issues.

## Decision Wrap Up:

1. Core libraries (or as Tyler more aptly put – EdgeX domain and common libraries) will exist and be maintain by Core/Supporting WG.
2. These should avoid any tight coupling to any technology (like DB).
3. The Application and Device Services can use at their option. If not used, they are responsible for adhering to the APIs/Models as determined by the spec (RAML, etc) and as reflected in the EdgeX common libraries (as represented by core libraries today).
4. All this should be documented on Wiki under architecture area. **Jim’s action item**
5. Architecture area should also include section to list additional questions/considerations to be undertaken by WG. **Jim’s action item**

**Multiple or Single Repos**

* Pros/Cons
* Different per language?

Janko – for mainflux moving to monrepo better visibility and easier to use.

Created separated folders for micro services. So no collisions

There could be issues when you want to mix Go services and Java services.

*Maybe make sense – mono report for particular layers (device services, app, core)?*

Tony – sounds reasonable to me

Repos language specific

For Go – layer repos

All agreed to try this out. Leave existing Java code as is for now.

**Messaging technology**

* Immediate needs between Core and Export services: what are the appropriate options to avoid Java/Go issuses
* Longer term, what should be considered the leading service to service communications protocols (outside of REST) and why?
* No time left to discuss. Left for next meeting

**Additional Open Architecture Questions**

* Tony – database specific – mongo db in API today. ID issue (name and separate ID) issue
* 2nd – how do we identify things like service
* These issues will be carried over to the next meetings

**Items considered Closed for now**

* Copyright header – stays the same until LF has an alternative
* Enums & JSON – use strings in the enums and serialize/deserialize as necessary in your language of choice.

Text copied from Chat Area

From Tyler\_Cox to Me: (Privately) 10:42 AM

It occurs to me that the root of the problem is that using core domain is considered "cross domain". I think the "proper" fix for this is to create a project domain that the layer domains inherit from, thus taking the project level object out of core domainthus isolating objects in the core layer from common project level objects

From Drasko - Mainflux to Everyone: 11:00 AM

I would go with with monorepo per WGfor exmple all core services can be in one repoand export services can be in other repoand device services can be in 3rd repoyesthis is how we structured Go Export servicesDo not mix Java nad go - specially for CII am in the bus, sorry :)This is the structure: https://github.com/drasko/edgex-exportthanks, bye