1. **General** 
   1. all log entries of Loglevel INFO or higher have unique, completely annotated PAI Message-ID
   2. all parameters of application configuration are defined as constants and annotated with@ConfigurationValue
   3. all IFs, IIFs and AFs are annotated
   4. all Collections (Sets, Listen, Arrays, Maps) are initialized when declared (?? Willburger, Dominik (059-Extern) it causes PMD/FindBugs violations about dead tore)
   5. Collection< ? > is used instead of implementation e.g. Set, List etc. (LB: This is most important for input parameters. A discussion about return values e.g. from persistance layer is already documented.)
   6. code is readable and divided into small understandble pieces, methods and variables have meaningfull names (LB: "small pieces" is very vague. Can we specify this further?)
   7. if some complex code is created then it is commented accordingly to allow future understanding of its functionality
   8. as long as it is possible there is no nested loops better performance, better readability, complex loops can oft be replaced by usage of maps
   9. where possible helper classes have been used StringUtils.isNotBlank StringUtils.join CollectionUtils.isNotEmpty , , , IOUtils ... (LB: We have a lot of own generic helper classes, as well)
   10. Streams and other resources are being closed e.g. IOUtils.closeQuietly dates are given with time zone UTC new DateTime(DateTimeZone.UTC) (?? Willburger, Dominik (059-Extern) is it up to date? in code all dates are created with time zone DE)
   11. all comments are complete and understandable (LB: I had many reviews in the near past where the comment did not match the code at all. -> Comments match the code.)
   12. not used code is being removed (not commented out) there is git for that
   13. all TODO/REFACTORME/FIXME tasks have been either incorporated or addressed and commented by corresponding JIRA-ID of the story/bug
   14. all input values and return values of Bean on layers over PS are logged with corrresponding log entry
   15. all loop and if conditions are readable and as simple as possible (LB: There are many points about "readability, small pieces, simplicity,complexity ...". Is it just KISS?)
   16. there is no spaghetti code, but each method usage is extracted to variable to allow better traceablity and debugging e.g. no return services.getService().getLicences().getCurrent().getId()
   17. code does not generate any FindBugs/PMD/Checkstyle violations
2. **Architecture** 
   1. all methods resides in correct interface

2.1.1 IifBs, IfBs and UiBs in Api-Project (without Impl suffix)

2.1.2 AfBs in Impl-Project - contains only AFs, for which there is no corresponding IifBs, IfBs or UiBs

* 1. call hierarchy is followed Bs Bf Ps (LB: Inside a component. Calling other components is only allowed for Bs -> Bs.)
  2. Bf recieves from Bs all data required by it from other components, and accesses only the data it is self responsible for JAVA-classes are stored in src/main/java (src/test/java for Test) and other resources are stored in src/main/resources (src/test/resources for Test)
  3. (LB: Reasoning about where to map BEs to DTOs, usage of TOs, ...)

1. **Database scripts** 
   1. BE has to mirror the database tables (see also mvn profile GenerateDDLs), this includes especially:

3.1.1 NOT NULL-Constraints

3.1.2 Default values

3.1.3 unique constraints incl. name of the constraint

* 1. all CREATE TABLE statements contain table space assignments
  2. all Primary Keys, Unique Constraints, Foreign Key Constraints and Indices have unique name
  3. internal foreign keys and foreign keys between components are defined in different initial sql files
  4. there are reasonable indices defined, especially for foreign keys
  5. transactional data inherits from AbstractOptimisticLockingEntity
  6. archive tables (\_H) implement ChangeTracked
  7. historical table have no constraints
  8. all CollectionTables are archived entity implements ChangeTrackedWithSpecificColumns or ChangeTrackedWithSpecificCollection
  9. initial- and delta- scripts are consistent
  10. DML delta scripts are able to be run multiple times without erros, especially:

MERGE or INSERT ... WHERE NOT EXISTS were used

* 1. database scripts are ready for work in high-availability environment (uninterruptible), and thus following operations are not being used, as they require DB REORG

3.12.1 SET NOT NULL

3.12.2 DROP NOT NULL

3.12.3 DROP COLUMN

* 1. v900 cleanup scripts have been created
  2. all constraints esp. FK on renamed tables have been recreated
  3. script name matches current sprint (LB: And is mentioned on the confluence page for delta scripts.)
  4. if SYSIBM queries are required then all contain condition containing CURRENT\_SCHEMA
  5. work packages for batches contain no foreign constraints on master or transactional data
  6. manual scripts contain set for PDFM\_MODIFICATIONDATE with current timestamp

UPDATE ..... set PDFM\_MODIFICATIONDATE = current timestamp

* 1. **Entities**
  2. technical primary key are always called id (without Prefix or Suffix). Corresponding getter is named getTechnicalId. Business key are called by business meaning e.g. modelSeriesId.
  3. are fields are accessed by property (annotations in getter)
  4. there is no direct access to fields of the entity only by corresponding getters also in the entity itself
  5. (LB: Collections have to be copied instead of using the direct reference which allows data manipulation.)

1. **Interfaces** 
   1. Asynchron methods are added to DispatcherBfBean.ASYNC\_OPERATIONS
   2. generated JAXB classes are used ionly in Dispatcher and Caller
   3. when creating ReST-interface access richts have been assigned according to SOE Service Security and for SOEWebServiceUserRole (old SOE Role)
2. **Dialogs** 
   1. data are taken from Change Session access only through FacadeBs, if necessary by a CsDataRetrieveStrategy
   2. dialog elements from SOE-DLG-Foundation are being used and if applicable extended change\_session\_button, paginator, translation\_256, ...
   3. Helper classes are being used if applicable and if required extended ComparatorHelper DateHelper I18NTextComparator, , , ... (LB: Abstract classes are used as in other dialogs: e.g. AbstractActionManager).
   4. Details in Change Session - Dialog Implementation.
3. **JUnit-Tests** 
   1. all test cases are commented, so that it is clear what is being done and why
   2. test data is as simple as possible there exist no not required test data, existiong data sets are duplicated and adapted instead of extended access to other components or layers is mocked
   3. XML Datasets for DB-Unit are used only in JUnit-Tests of PS
   4. classes from testdata-Packages are used
   5. test data from testdata-Packages are rolledback/reseted
   6. there is a test case for each new bug fix
   7. where applicable corresponding jUnit rules have been used ExpectedException, SystemPropertiesRule
   8. test cases follow //GIVEN // WHEN //THEN convention
   9. test cases are defined for both positive and negative test cases
   10. @Ignore is always with corresponding comment about reasons, and JIRA-ID of task that should correct the situation
   11. test Methods use onlt throws Exception instead of any other detail exceptions like i.e. SoeBusinessException
4. **SOAP-UI Tests** 
   1. Smoke test for REST Service has been created
   2. Smoke test contains example request
   3. Smoke test contains schema validation
   4. Smoke test contains correct value of property Name (incl. version) in Test Positive Cases suite
   5. Name of test suite matches the convention e.g. Test Cases - Services\_getUserAgreementMasterDataForCountry
5. **Dead code** 
   1. there is no commented out code for future use
   2. all private and public methods are being called by productive code

By dead code is meant a code that is no more used by any other part of the system. It can happen during big changes like refactoring where old or no more required interface will be removed. It is hard to find such code snippets especially if they are public and is possible that they will be missed initially.If any such code will be found then it should be either removed directly or if the amount of work can increase should be marked by REFACTORME task in code and corresponding JIRA-Task should be created

(LB: Some code is accessed using reflection. Removing dead code should only be done in case it becomes obsolete due to new requirements.)

1. **Javadoc** 
   1. general @author Capgemini tag has been used in JavaDoc
   2. all public methods have JavaDoc
   3. each Java class contains Daimler Header (LB: Using the current year, File Name, File Type, ...)
2. **Naming Conventions** 
   1. Named Queries and corresponding constants are SQL-Statement oriented e.g. QUERY\_SELECT\_ALL\_BUSINESS\_KEYS = "Ps.selectAllBusinessKeys".
   2. method names come from CRUD database operations (create, read, update, delete), e.g. Ps.createBusinessKey(), Ps.readBusinessKey(), Ps.readBusinessKeyByCountry(String country), Ps.readAllBusinessKeys(), Ps.readAllBusinessKeysByCountry(String country).
   3. method names in Bf- and Bs- layer (incl. DTOs) however are object oriented, e.g. Bs.addBusinessKey(), Bs.getAllBusinessKeys().