TABLE I

General Annotation Policies Codes and Categories

ID	Code	Category
OSS_3	Code should be self documenting.	Never annotate
OSS_{17}	codigo fuente y diseno	
OSS_{23}	Code comments should be avoided. Code is documented by itself and appropriate unit tests. Following clean code principles and investing a lot of time into good naming of variables and functions is key.	
Ind_{19}	In principle the code should be clean and clear such that no additional annotations are needed to understand the code.	
Ind_{24}	Our organization has just a traditional clean code policy	
Ind_{27}	Following clean code principles and investing a lot of time into good naming of variables and functions is key.	
Ind_{32}	The agreement is that the code should speak for itself and if the design needs explanation then it's probably wrong and needs to go back to the drawing board.	
Ind_{33}	Code, combined with unit tests should be self-explanatory. If comments are needed to understand the code, that is a code smell.	
Ind_{40}	But if a pull request is reviewed and the reviewing programmer does not understand the code, then the reviewer will ask to simplify the code, or if that is not easily possible, add comments for the cr process.	
OSS_6	Most high level classes are expected to be documented, specially how they fit into the whole picture, semantics around caller usage, thread-safety etc.	Always annotate
OSS_{44}	I'm currently on a project where 10% of each day is used to document the days work. Each method should have a java doc comment and each inline variable declaration should have a normal comment describing what it is going to be used for. Every loop should also have a normal comment above its declaration describing a typical execution scenario.	
Ind_6	A Public procedure must be tested and documented.	
Ind_{19}	Interfaces should be fully documented (how to use the code).	
Ind_{23}	if the code is not annotated enough, your colleagues/superiors will complain in person	
Ind_{24}	policy with high endorsement of using annotations.	
Ind_{49}	Our coding standard makes it mandatory to describe/annotate every record, function signature and package	
OSS_2	When we feel something isn't straight fo[r]ward (or when in doubt), we always add some code comments.	Annotate under special circumstances
OSS_3	When something is complicated it should be commented.	
OSS_7	Explanations are fine for more complex choices.	
OSS_{25}	Eslint restrictions, not using case insensitive names for components etc.	
OSS_{28}	My policy is always to document any unusual design/implementation choices, or those that might be reusable in the future	
OSS_{30}	https://www.tensorflow.org/community/contribute/code_style	
Ind_5	we have a high level design document. autogenerated api docs such as doxygen, sphinx, or godoc.	
Ind_6	Tricky implementations details must made public by a subpackage tested and documented as internals.	
Ind_{27}	Use comments only if you need to explain something which is not immediately understandable by reading the code.	
Ind_{32}	However real life sometimes calls for exceptions, so you can defend and explain your need of annotations during code reviews and the team can raise suggestions or ideas on the topic at hand.	
Ind_{34}	However as a general rule any comment added to the source code should be meaningful. *i.e.,* no obvious remarks or commenting old code when making design changes	
Ind_{40}	If comments are superfluous, a reviewer can ask to remove the comment, as it just distracts.	
Ind_{46}	Informal ones, to be peer-reviewed for adequacy	
Ind_{47}	Yes, where necessary comments should be added, but this is not closely monitored	
OSS_9	Yes, an architecture document should be created for any medium to large sized features describing the architecture and rationale.	Describe high-level design decisions
OSS_{11}	Rarely, design and development quirks/choices to be collected in Wiki documentation	
OSS_{37}	Generally a dedicated docs directory with a per-subsystem document outlining the high level design and documented interfaces for easy navigation in an IDE.	
OSS_{38}	Github wiki / markdown files for conceptual overview; Directory for notebooks for initial research and testing (code not used in production, but instructive for extensions and future reference).	
OSS_{40}	[anonymized] has both user documentation and detailed peer review discussions on GitHub.	
OSS_{42}	Other than writing design docs for major features, no.	
OSS_{46}	design choices are documented outside code in separate document	
Ind_5	we have a high level design document. autogenerated api docs such as doxygen, sphinx, or godoc.	
Ind_{21}	We try to document high level design choices in the README.md	
Ind_{22}	Not in code. Though architecture decisions on company level are captured with ADRs.	
Ind_{40}	Also, if a comment can be put into a format like JavaDoc, then use JavaDoc, as you can then generate a nice HTML page for it.	
Ind_{45}	Design choices have to be documented in the model of the SW architecture or detailed design.	
Ind_{53}	Yes, doxygen style.	
OSS_{41}	I am a maintainer of [anonymized]. Together with [anonymized] we follow the guidelines given in [anonymized] and many more probably unwritten protocols.	Link to issue trackers
Ind_{19}	Extra annotation is done through the commit messages: each commit message must contain a reference to the issue tracking system, such that it is always clear in which context some code was changed.	
OSS_5	Best practices' are specified, but rarely defined. When they are they result in a maze of links which discourages reading them.	Avoid link maze
OSS_8	it depends on the developer and the cr reviewer	Decisions left to the team
OSS_{42}	We have no policies on in-line comments in source code. It is up to the developer to decide.	
Ind_9	No. Everything is team dependent.	
Ind_{19}	No policy within the organization, but sometimes a policy within the team/project	
Ind_{40}	Proactively adding comments depends on the individual programmer.	

TABLE II
SATD ANNOTATION POLICIES CODES AND CATEGORIES (OSS)

ID	Code	Category
OSS$_1$	FIXME must be fixed before release, preferably before commit.	Never annotate
OSS$_{38}$	We have an automated git hook that checks no FIXMEs are present in merged code.	
OSS$_6$	No, but we typically do TODO annotations [in] the source code so that they are not confused.	Document as comments
OSS$_{13}$	Visual studio has TODO and HACK functionality build in, so I do use both.	
OSS$_{19}$	Only simplistic ones: use capitals in TODO/FIXME.	
OSS$_{40}$	ad-hoc todo code comments	
OSS$_{12}$	We utilize the annotation # HACK: for highlighting TD workarounds within our private source code.	Do not document in code
OSS$_1$	All markers must be annotated by the developer user name.	Declare your identity
OSS$_1$	TODO is a reminder and may be moved to an issue in the tracker	Link to issue tracker
OSS$_2$	The policy is to create tasks on our issue tracker	
OSS$_3$	We have an issue tracking system where we manage things that need doing.	
OSS$_7$	generally future work shouldn't be documented in code instead in tickets.	
OSS$_{23}$	Tech-Debt, are documented using a ticket in our jira or github issue tracker.	
OSS$_{37}$	Some ban TODO style comments and require tickets to be raised	
OSS$_{41}$	The typical keyword "[WIP]" denoting work-in-progress is put in the Pull Requests on GitHub repo.	
OSS$_{42}$	For TODOs: Prefer specifying a issue id over a username.	
OSS$_{44}$	We use other means of keeping track of this (GitLab issues)	
OSS$_{32}$	TODO (automatically recognized by PyCharm)	Use tool support
OSS$_{16}$	We also utilize a "# TODO: " annotation internally, but no formal policies regarding annotations in general.	Decisions left to the team
OSS$_{34}$	I generally use TODO.	
OSS$_{43}$	No, but I have my own policies: TODO(issue12313) to link to a bug or something, and only if I am actually going to do it.	

TABLE III
SATD ANNOTATION POLICIES CODES AND CATEGORIES (IND)

ID	Code	Category
Ind$_{13}$	Our CICD workflow prevents TODO, Fixme, etc comments from proceeding past the DEV branch.	Never annotate
Ind$_{19}$	MUDO: this is a todo that really must be done before putting the code in production	
Ind$_{28}$	Not written, but are strongly discouraged.	
Ind$_{32}$	FIXMEs need to be fixed before merging into the main branch and/or releasing.	
Ind$_{33}$	They should never be used.	
Ind$_{45}$	No, delivered code shall be free from annotations	
Ind$_{50}$	You can't push code to develop branches with TODO's or FIXME's.	
Ind$_{52}$	No hard policies, but it is far from preferred.	
Ind$_4$	They should always be documented	Document as comments
Ind$_{19}$	Inline comments on the code are only needed if the code is complex	
Ind$_{32}$	when absolutely needed, they are usually explained not in code but in documents that are in our knowledge base	Do not document in code
Ind$_{34}$	No pragma messages in source code to force todo messages to be visible in the compiler.	
Ind$_{18}$	we usually specify TODOs as: TODO: Name: Explanation.	Declare your identity
Ind$_{21}$	no TODO's without names or initials of the engineer.	
Ind$_{32}$	it is expected to attach a name to a TODO so that it is visible from code without git-blaming	
Ind$_{42}$	Preferably with name of author, date, and rationale.	
Ind$_{49}$	a convention for TODOs: "TODO (%svn username%)".	
Ind$_{53}$	Yes, notation is either //TODO or //FIXME, followed by date, developer who wrote this and only after that the explanation.	
Ind$_6$	TODO must be referenced as an issue number in the TODO comment.	Link to issue tracker
Ind$_{33}$	There are policies of traceability of known issues.	
Ind$_{34}$	An issue should be created in the issue-tracker instead.	
Ind$_{40}$	A TODO should always be accompanied by a task nr, in our case the JIRA task ID.	
Ind$_{49}$	We also often add issues for technical debts in Jira, following a review.	
Ind$_{51}$	We also often add issues for technical debts in Jira, following a review.	
Ind$_{53}$	If possible, related ticket ID.	
Ind$_{21}$	The code is scanned by Sonar and warns you that you're committing a todo.	Use tool support
Ind$_{24}$	There are also automated code and requirement robot tests that check if specifications changes were followed cross-release with or without carried over work.	
Ind$_{19}$	No policy within the organisation, but sometimes a policy within the team/project.	Decisions left to the team
Ind$_{24}$	Tech leads of dev, test and ops sometimes follow different implementation and design practices depending on tools and frameworks.	