Ministerie van Binnenlandse Zaken en Koninkrijksrelaties

Versie 1.3

Ministerie van BZK
Operatie BRP

Turfmarkt 147
2511 DC Den Haag
Postbus 10451
2501 HL Den Haag

www.operatieBRP.nl
info@operatiebrp.nl

notitie

Datum
12 april 2016

Inleiding
Het doel van deze notitie is de aanpassing van het normenkader codekwaliteit Centrale BRP voorzieningen (versie 1.2 vastgesteld door stuurgroep Operatie BRP op 14 april 2016) aan te geven. Aanleiding voor het aanpassen van het normenkader is het advies van KPMG aan de stuurgroep om te voldoen aan de industrie standaard (Register aanbevelingen QA, nr. 25b). Deze notitie geeft de aanpassingen tussen het vastgestelde normenkader en het nieuwe normenkader weer. Voor de volledigheid is het nieuwe normenkader opgenomen.
Deze versie van het normenkader wordt sinds 7 december 2016 reeds toegepast. Het nieuwe kader is door KPMG getoetst. In de stuurgroep van 20 april 2017 wordt daarover gerapporteerd.

Aanpassingen in de normstelling
De vastgestelde normen (versie 1.2 door stuurgroep Operatie BRP op 14 april 2016) zijn weergegeven in onderstaande tabel.

Nr.	Aspect	Norm
1	Aantal blocker of critical issues	0 (nul, geen)
2	Aantal issues ten aanzien van veiligheid en betrouwbaarheid	0 (nul, geen)
3	Testdekking op productiecode	Minimaal 80%
4	Documentatie publieke API	Minimaal 95%
5	Code duplicatie	Maximaal 4%
6	Cyclische afhankelijkheden	0 (nul, geen)
7	Rule Compliance Index	Minimaal 97%

In overleg met KPMG is gekozen om de industriestandaard voor de regelset te interpreteren als de standaard regelset van Sonar. De standaard regelset van Sonar (Sonar Way) vormt de basis van de regelset voor Operatie BRP. Sonar is m.b.t. de regelset een andere weg ingeslagen en hanteert nu de indeling: bugs, vulnerabilities en code smells. Dat betekent dat 2 normen aangepaste benamingen krijgen. In onderstaande tabel zijn de wijzigingen gearceerd.

Nr.	Aspect	Norm
1	Aantal blocker of critical issues **voor code smells**	0 (nul, geen)
2	Aantal issues ten aanzien van **vulnerabilities en bugs**	0 (nul, geen)

3	Testdekking op productiecode	Minimaal 80%
4	Documentatie publieke API	Minimaal 95%
5	Code duplicatie	Maximaal 4%
6	Cyclische afhankelijkheden	0 (nul, geen)
7	Rule Compliance Index	Minimaal 97%

De te hanteren meetmethode per aspect is onderdeel van de norm, en is beschreven in bijlage 1 aan het eind van dit document.

De grootste wijziging die plaatsgevonden heeft, is de regelset. Aangezien Sonar de regels grotendeels gewijzigd heeft, is de delta niet weer te geven. De regelset bevat 3 categorieën:

1. Sonar Way[1]
2. Java formatting
3. Operatie BRP regels (aanvullende regels om de software onderhoudbaar, betrouwbaar en veilig te maken)

Deze profielen zijn gebaseerd op de regels van de SonarQube Java plugin versie 4.2.1.6971 en de SonarQube XML plugin versie 1.4.1.

In bijlage 2 zijn de nieuwe regels weergegeven.

Afbakening

De kwaliteitsnorm is integraal van toepassing op code die door project O&R zelf is geschreven en die bedoeld is om in productie te gaan. De volgende code wordt buiten beschouwing gelaten:

- third party libraries (hier onder worden ook (her)gebruikte delen van bestaande GBA-V broncode verstaan) alsmede code gegenereerd door deze libraries.

[1] Bij de Sonar Way is 1 uitzondering gemaakt en betreft de regel:

(FunctionalInterface annotation should be used to flag Single Abstract Method interfaces)

Een functionalInterface annotatie legt intentie vast dat een interface ook echt bedoeld is als functionele interface. Een interface met 1 methode is technisch gezien altijd een functional interface, maar dit betekent niet dat het de intentie van de programmeur was om deze interface een functionele interface te laten zijn. Als dat wel de intentie was moet een annotatie geplaatst worden. Het afdwingen van de functional interface annotatie leidt tot foute markeringen. Het plaatsen van de functional interface is uiteindelijk een ontwerpkeuze waarvan Sonar niet kan bepalen of die correct is.

Bijlage 1: Meetmethoden

Nr	Aspect	Meetmethode
1	Aantal blocker of critical issues voor code smells	In SonarQube worden de metrics: blocker en critical issues gebruikt. De complete regelset is opgenomen in bijlage 2.
2	Aantal issues ten aanzien van vulnerabilities en bugs	In SonarQube worden de metrics: vulnerabilities en bugs gebruikt. De complete regelset is opgenomen in bijlage 2.
3	Testdekking op productiecode	De norm geldt voor zowel line coverage als branch coverage. Er wordt in het buildproces gebruik gemaakt van JaCoCo maven plugin versie 0.7.9. In SonarQube worden de metrics: condition coverage en line coverage gebruikt.
4	Documentatie publieke API	In SonarQube wordt de metric: public documented API (%) gebruikt.
5	Code duplicatie	In SonarQube wordt de metric: duplicated line (%) gebruikt.
6	Cyclische afhankelijkheden	In SonarQube wordt de metric: package dependency cycles gebruikt. Hiervoor wordt gebruik gemaakt van de jDepend Plugin for SonarQube versie 1.0
7	Rule Compliance Index	De RCI wordt berekend aan de hand van de metrics: blocker, critical, major en minor issues uit SonarQube. Gewogen aantal issues = 10 × aantal blocker issues + 5 × aantal critical issues + 3 × aantal major issues + aantal minor issues RCI =(1– (gewogen aantal issues/aantal regels code)) × 100% Er wordt gebruik gemaakt van de plugin Rule Compliance Index (RCI) versie 1.0.1.

"Explains" maken geen deel uit van de telling ten behoeve van de normen 1 tot en met 6. Ten aanzien van norm 7 wordt de specifieke bijdrage van de issues die de "Explains" veroorzaken in de berekening van de Rule Compliance Index in kaart gebracht, de berekening van de index wordt voor deze bijdrage gecorrigeerd door de door "Explains" veroorzaakte bevindingen in de telling mee te nemen.

Norm nummer 7 is erop gericht om te borgen dat het totaal aantal issues (inclusief het deel met een 'explain') proportioneel blijft ten opzichte van de omvang van de code.
In dit normenkader worden uitsluitingen van aangewezen delen van de code voor specifieke normen vastgelegd. Aangewezen delen betreffen de stukken code zoals benoemd in de afbakening en de markeringen ten behoeve van de "explains". Deze uitsluitingen worden door middel van een technische markering in de code aangebracht ten behoeve van de gebruikte tooling. Deze technische markering kan op zichzelf ook leiden tot een afwijking van een regel. Deze markeringen tellen niet mee bij tellingen ten behoeve van alle normen (inclusief norm 7) en behoeven geen specifieke "explain". De algemene verantwoording is het technisch mogelijk maken van metingen van de codekwaliteit op basis van het normenkader.

Bijlage 2: Regels

Bugs

- ".equals()" should not be used to test the values of "Atomic" classes
- "@NonNull" values should not be set to null
- "Arrays.stream" should be used for primitive arrays
- "assert" should only be used with boolean variables
- "BigDecimal(double)" should not be used
- "Cloneables" should implement "clone"
- "compareTo" results should not be checked for specific values
- "compareTo" should not return "Integer.MIN_VALUE"
- "ConcurrentLinkedQueue.size()" should not be used
- "DateUtils.truncate" from Apache Commons Lang library should not be used
- "deleteOnExit" should not be used
- "Double.longBitsToDouble" should not be used for "int"
- "entrySet()" should be iterated when both the key and value are needed
- "equals" methods should be symmetric and work for subclasses
- "equals(Object obj)" and "hashCode()" should be overridden in pairs
- "equals(Object obj)" should be overridden along with the "compareTo(T obj)" method
- "equals(Object obj)" should test argument type
- "Exception" should not be caught when not required by called methods
- "Externalizable" classes should have no-arguments constructors
- "for" loop incrementers should modify the variable being tested in the loop's stop condition
- "hashCode" and "toString" should not be called on array instances
- "indexOf" checks should not be for positive numbers
- "instanceof" operators that always return "true" or "false" should be removed
- "InterruptedException" should not be ignored
- "Iterator.hasNext()" should not call "Iterator.next()"
- "Iterator.next()" methods should throw "NoSuchElementException"
- "main" should not "throw" anything
- "notifyAll" should be used
- "null" should not be used with "Optional"
- "Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop
- "PreparedStatement" and "ResultSet" methods should be called with valid indices
- "read" and "readLine" return values should be used
- "ResultSet.isLast()" should not be used
- "runFinalizersOnExit" should not be called
- "ScheduledThreadPoolExecutor" should not have 0 core threads
- "Serializable" inner classes of non-serializable classes should be "static"
- "SingleConnectionFactory" instances should be set to "reconnectOnException"
- "StringBuilder" and "StringBuffer" should not be instantiated with a character

- "super.finalize()" should be called at the end of "Object.finalize()" implementations
- "switch" statements should not contain non-case labels
- "toArray" should be passed an array of the proper type
- "toString()" and "clone()" methods should not return null
- "toString()" should never be called on a String object
- "URL.hashCode" and "URL.equals" should be avoided
- "wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held
- "wait(...)", "notify()" and "notifyAll()" methods should only be called when a lock is obviously held on an object
- A "for" loop update clause should move the counter in the right direction
- Blocks should be synchronized on "private final" fields
- Boxing and unboxing should not be immediately reversed
- Case insensitive string comparisons should be made without intermediate upper or lower casing
- Child class methods named for parent class methods should be overrides
- Classes and methods that rely on the default system encoding should not be used
- Classes extending java.lang.Thread should override the "run" method
- Classes should not be compared by name
- Classes that don't define "hashCode()" should not be used in hashes
- Classes that override "clone" should be "Cloneable" and call "super.clone()"
- Collection.isEmpty() should be used to test for emptiness
- Collections should not be passed as arguments to their own methods
- Conditions should not unconditionally evaluate to "TRUE" or to "FALSE"
- Constructor injection should be used instead of field injection
- Constructors should not be used to instantiate "String" and primitive-wrapper classes
- Custom serialization method signatures should meet requirements
- Dead stores should be removed
- Dependencies should not have "system" scope
- Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting
- Double Brace Initialization should not be used
- Empty statements should be removed
- Exception handlers should preserve the original exceptions
- Exceptions should not be thrown in finally blocks
- Exit methods should not be called
- Fields in a "Serializable" class should either be transient or serializable
- Floating point numbers should not be tested for equality
- Generic exceptions should never be thrown
- Getters and setters should be synchronized in pairs
- Identical expressions should not be used on both sides of a binary operator
- Inappropriate "Collection" calls should not be made
- Inappropriate regular expressions should not be used
- Inner classes which do not reference their owning classes should be "static"

- Ints and longs should not be shifted by zero or more than their number of bits-1
- Invalid "Date" values should not be used
- Java 8's "Files.exists" should not be used
- Jump statements should not occur in "finally" blocks
- Lazy initialization of "static" fields should be "synchronized"
- Locks should be released
- Loop conditions should be true at least once
- Loops should not be infinite
- Maps with keys that are enum values should be replaced with EnumMap
- Math operands should be cast before assignment
- Math should not be performed on floats
- Method parameters, caught exceptions and foreach variables should not be reassigned
- Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances
- Methods named "equals" should override Object.equals(Object)
- Methods of "Random" that return floating point values should not be used in random integer generation
- Methods should not be named "hashcode" or "equal"
- Modulus results should not be checked for direct equality
- Multiline blocks should be enclosed in curly braces
- Multiple loops over the same set should be combined
- Neither "Math.abs" nor negation should be used on numbers that could be "MIN_VALUE"
- Non-public methods should not be "@Transactional"
- Non-serializable classes should not be written
- Non-serializable objects should not be stored in "HttpSession" objects
- Non-thread-safe fields should not be static
- Null pointers should not be dereferenced
- Objects should not be created only to "getClass"
- Optional value should only be accessed after calling isPresent()
- Parsing should be used to convert "Strings" to primitives
- Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls
- Primitives should not be boxed just for "String" conversion
- Printf-style format strings should not lead to unexpected behavior at runtime
- Raw byte values should not be used in bitwise operations in combination with shifts
- Reflection should not be used to check non-runtime annotations
- Related "if/else if" statements should not have the same condition
- Resources should be closed
- Return values should not be ignored when function calls don't have any side effects
- Servlets should not have mutable instance fields
- Sets with elements that are enum values should be replaced with EnumSet
- Short-circuit logic should be used in boolean contexts
- Silly bit operations should not be performed

- Silly equality checks should not be made
- String function use should be optimized for single characters
- Strings literals should be placed on the left side when checking for equality
- Strings should not be concatenated using '+' in a loop
- Subclasses that add fields should override "equals"
- Switch cases should end with an unconditional "break" statement
- Synchronization should not be based on Strings or boxed primitives
- Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used
- The non-serializable super class of a "Serializable" class should have a no-argument constructor
- The Object.finalize() method should not be called
- The Object.finalize() method should not be overriden
- The signature of "finalize()" should match that of "Object.finalize()"
- The value returned from a stream read should be checked
- Thread.run() should not be called directly
- Throwable and Error should not be caught
- Two branches in the same conditional structure should not have exactly the same implementation
- Value-based objects should not be serialized
- Values should not be uselessly incremented
- Variables should not be self-assigned
- Zero should not be a possible denominator

Vulnerabilities

- "enum" fields should not be publicly mutable
- "File.createTempFile" should not be used to create a directory
- "HttpServletRequest.getRequestedSessionId()" should not be used
- "javax.crypto.NullCipher" should not be used for anything other than testing
- "public static" fields should be constant
- Class variable fields should not have public accessibility
- Classes should not be loaded dynamically
- Cookies should be "secure"
- Credentials should not be hard-coded
- Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding)
- Default EJB interceptors should be declared in "ejb-jar.xml"
- Exceptions should not be thrown from servlet methods
- HTTP referers should not be relied on
- IP addresses should not be hardcoded
- Mutable fields should not be "public static"
- Neither DES (Data Encryption Standard) nor DESede (3DES) should be used
- Only standard cryptographic algorithms should be used
- Pseudorandom number generators (PRNGs) should not be used in secure contexts
- Return values should not be ignored when they contain the operation status code

- SHA-1 and Message-Digest hash algorithms should not be used
- SQL binding mechanisms should be used
- Throwable.printStackTrace(...) should not be called
- Untrusted data should not be stored in sessions
- Values passed to LDAP queries should be sanitized
- Values passed to OS commands should be sanitized
- Web applications should not have a "main" method
- Web applications should use validation filters

Code smells

- "@Deprecated" code should not be used
- "@Override" should be used on overriding and implementing methods
- "action" mappings should not have too many "forward" entries
- "catch" clauses should do more than rethrow
- "clone" should not be overridden
- "Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used
- "final" classes should not have "protected" members
- "finalize" should not set fields to "null"
- "for" loop stop conditions should be invariant
- "indexOf" checks should use a start position
- "java.lang.Error" should not be extended
- "Lock" objects should not be "synchronized"
- "NullPointerException" should not be caught
- "Object.finalize()" should remain protected (versus public) when overriding
- "Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"
- "Optional" should not be used for parameters
- "private" methods called only by inner classes should be moved to those classes
- "readObject" should not be "synchronized"
- "readResolve" methods should be inheritable
- "Serializable" classes should have a version id
- "Serializable" inner classes of "Serializable" classes should be static
- "static" members should be accessed statically
- "switch case" clauses should not have too many lines
- "switch" statements should end with "default" clauses
- "switch" statements should have at least 3 "case" clauses
- "switch" statements should not have too many "case" clauses
- "Thread.sleep" should not be used in tests
- "Threads" should not be used where "Runnables" are expected
- "throws" declarations should not be superfluous
- A "while" loop should be used instead of a "for" loop
- A close curly brace should be located at the beginning of a line
- A field should not duplicate the name of its containing class
- Abstract class names should comply with a naming convention
- Abstract classes without fields should be converted to interfaces
- Abstract methods should not be redundant
- An abstract class should have both abstract and concrete methods
- An open curly brace should be located at the end of a line
- Annotation repetitions should not be wrapped
- Anonymous inner classes containing only one method should become lambdas

- Array designators "[]" should be located after the type in method signatures
- Array designators "[]" should be on the type, not the variable
- Artifact ids should follow a naming convention
- Assertions should be complete
- Assignments should not be made from within sub-expressions
- Boolean checks should not be inverted
- Boolean literals should not be redundant
- Branches should have sufficient coverage by tests
- Catches should be combined
- Child class fields should not shadow parent class fields
- Class names should comply with a naming convention
- Class names should not shadow interfaces or superclasses
- Classes from "sun.*" packages should not be used
- Classes named like "Exception" should extend "Exception" or a subclass
- Classes should not be empty
- Classes with only "static" methods should not be instantiated
- Classes without "public" constructors should be "final"
- Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line
- Collapsible "if" statements should be merged
- Comments should not be located at the end of lines of code
- Comparators should be "Serializable"
- Constant names should comply with a naming convention
- Constants should not be defined in interfaces
- Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply
- Control structures should use curly braces
- Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"
- Deprecated "${pom}" properties should not be used
- Deprecated code should be removed
- Deprecated elements should have both the annotation and the Javadoc tag
- EJB interceptor exclusions should be declared as annotations
- Empty arrays and collections should be returned instead of null
- Enumeration should not be implemented
- Equality operators should not be used in "for" loop termination conditions
- Escaped Unicode characters should not be used
- Exception classes should be immutable
- Exception types should not be tested using "instanceof" in catch blocks
- Execution of the Garbage Collector should be triggered only by the JVM
- Expressions should not be too complex
- Field names should comply with a naming convention
- Fields in non-serializable classes should not be "transient"
- Files should contain an empty new line at the end
- Files should contain only one top-level class or interface each
- Files should not be empty
- Files should not have too many lines
- Future keywords should not be used as names
- Generic wildcard types should not be used in return parameters
- Group ids should follow a naming convention
- IllegalMonitorStateException should not be caught
- Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression

- Inheritance tree of classes should not be too deep
- Inner class calls to super class methods should be unambiguous
- Instance methods should not write to "static" fields
- Interface names should comply with a naming convention
- Java parser failure
- JUnit assertions should not be used in "run" methods
- JUnit framework methods should be declared properly
- JUnit rules should be used
- JUnit test cases should call super methods
- Labels should not be used
- Lambdas and anonymous classes should not have too many lines
- Lambdas should be replaced with method references
- Lamdbas containing only one statement should not nest this statement in a block
- Lines should not be too long
- Literal suffixes should be upper case
- Local variable and method parameter names should comply with a naming convention
- Local Variables should not be declared and then immediately returned or thrown
- Local variables should not shadow class fields
- Loggers should be "private static final" and should share a naming convention
- Loops should not contain more than a single "break" or "continue" statement
- Magic numbers should not be used
- Method names should comply with a naming convention
- Method overrides should not change contracts
- Methods and field names should not be the same or differ only by capitalization
- Methods should not be empty
- Methods should not be too complex
- Methods should not have too many lines
- Methods should not have too many parameters
- Methods should not have too many return statements
- Methods should not return constants
- Modifiers should be declared in the correct order
- Multiple variables should not be declared on the same line
- Nested "enum"s should not be declared static
- Nested blocks of code should not be left empty
- Nested code blocks should not be used
- Non-constructor methods should not have the same name as the enclosing class
- Null should not be returned from a "Boolean" method
- Octal values should not be used
- Only static class initializers should be used
- Overriding methods should do more than simply call the same method in the super class
- Package declaration should match source file directory
- Package Dependency Cycles
- Package names should comply with a naming convention
- Packages should have a javadoc file 'package-info.java'
- Parentheses should be removed from a single lambda input parameter when its type is inferred
- pom elements should be in the recommended order

- Private fields only used as local variables in methods should become local variables
- Public constants and fields initialized at declaration should be "static final" rather than merely "final"
- Public methods should throw at most one checked exception
- Public types, methods and fields (API) should be documented with Javadoc
- Redundant casts should not be used
- Redundant modifiers should not be used
- Return of boolean expressions should not be wrapped into an "if-then-else" statement
- Sections of code should not be "commented out"
- Silly math should not be performed
- Simple class names should be used
- Skipped unit tests should be either removed or fixed
- Source files should not have any duplicated blocks
- Standard outputs should not be used directly to log anything
- Statements should be on separate lines
- Static non-final field names should comply with a naming convention
- String literals should not be duplicated
- String.valueOf() should not be appended to a String
- Tabulation characters should not be used
- Test classes should comply with a naming convention
- TestCases should contain tests
- Tests should not be ignored
- The default unnamed package should not be used
- The diamond operator ("<>") should be used
- The members of an interface declaration or class should appear in a pre-defined order
- Threads should not be started in constructors
- Track uses of "FIXME" tags
- Track uses of "TODO" tags
- Try-catch blocks should not be nested
- Try-with-resources should be used
- Type parameter names should comply with a naming convention
- Underscores should be used to make large numbers readable
- Unnecessary semicolons should be omitted
- Unused "private" fields should be removed
- Unused "private" methods should be removed
- Unused labels should be removed
- Unused local variables should be removed
- Unused method parameters should be removed
- Unused type parameters should be removed
- Useless imports should be removed
- Useless parentheses around expressions should be removed to prevent any misunderstanding
- Utility classes should not have public constructors
- Wildcard imports should not be used

Bijlage 3: NFR's

De volgende NFR's worden volledig ingevuld door de norm:

Code	Requirement
RD-OH-001	De broncode voldoet aan codeerrichtlijnen en wordt geautomatiseerd getest op deze codeerrichtlijnen. De gehanteerde codeerrichtlijnen zijn representatief voor binnen het vakgebied gebruikelijke richtlijnen voor professionele softwareontwikkeling.
RD-BEV-001	95% van de methodes van publieke interfaces is gedocumenteerd.