

SONARQUBE

공개SW가 테스트를 만나다!
14⁴⁵-15³⁰/20150429/수
강남SC컨벤션12F아나이스홀

지속적인 통합 환경에서의
SW품질 시각화를 통한
효율적인 코드 리뷰 수행 방안

김모세

Quality Assurance Engineer@NBT Partners
SonarQube Korean Localization Lead



✉ creatinov.kim@gmail.com

📄 www.creatinov.org

📘 <https://www.facebook.com/groups/korea.sonarqube.user.group/>

FOCUS

- ✓ Code Quality
- ✓ Code Review
- ✓ Static Analysis
- ✓ SonarQube + CI

Code Quality

What is good code?

Bjarne Stroustrup

Inventor of C++ and author of *The C++ Programming Language*

*I like my code to **be elegant and efficient**. The logic should be **straightforward** to make it **hard for bugs to hide**, the **dependencies minimal** to **ease maintenance, error handling complete** according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. Clean code **does one thing** well.*



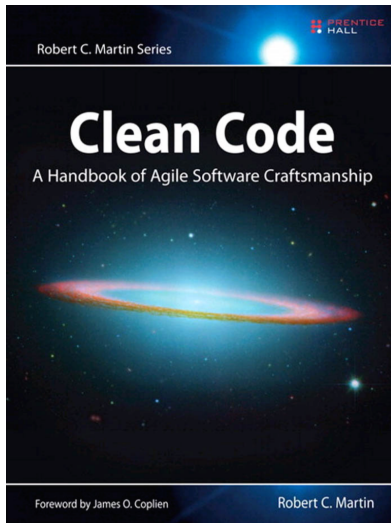
*You know you are working on clean code when **each routine you read turns out to be pretty much what you expected**. You can call it beautiful code when the code also makes it **look like the language was made for the problem**.*

Ward Cunningham

Inventor of Wiki/ Fit. Co-inventor of eXtreme Programming.

Motive force behind Design Patterns. Smalltalk and OO thought leader.

The godfather of all those who care about code



Contents	
Foreword.....	xix
Introduction.....	xxv
On the Cover.....	xxix
Chapter 1: Clean Code.....	1
There Will Be Code.....	2
Bad Code.....	3
The Total Cost of Owning a Mess.....	4
The Grand Redesign in the Sky.....	5
Attitude.....	5
The Primal Conundrum.....	6
The Art of Clean Code?.....	6
What Is Clean Code?.....	7
Schools of Thought.....	12
We Are Authors.....	13
The Boy Scout Rule.....	14
Prequel and Principles.....	15
Conclusion.....	15
Bibliography.....	15
Chapter 2: Meaningful Names.....	17
Introduction.....	17
Use Intention-Revealing Names.....	18
Avoid Disinformation.....	19
Make Meaningful Distinctions.....	20
Use Pronounceable Names.....	21
Use Searchable Names.....	22

Contents	
Avoid Encodings.....	23
Hungarian Notation.....	23
Member Prefixes.....	24
Interfaces and Implementations.....	24
Avoid Mental Mapping.....	25
Class Names.....	25
Method Names.....	25
Don't Be Cute.....	26
Pick One Word per Concept.....	26
Don't Put.....	26
Use Solution Domain Names.....	27
Use Problem Domain Names.....	27
Add Meaningful Context.....	27
Don't Add Gratuitous Context.....	29
Final Words.....	30
Chapter 3: Functions.....	31
Small!.....	34
Blocks and Indenting.....	35
Do One Thing.....	35
Function Arguments.....	40
Common Monadic Forms.....	41
Flag Arguments.....	41
Dyadic Functions.....	42
Triads.....	42
Argument Objects.....	43
Argument Lists.....	43
Verbs and Keywords.....	43
Have No Side Effects.....	44
Output Arguments.....	45
Command Query Separation.....	45

Contents	
Prefer Exceptions to Returning Error Codes.....	46
Extract Try/Catch Blocks.....	46
Error Handling Is One Thing.....	47
The Error-Java Dependency Magnet.....	47
Don't Repeat Yourself.....	48
Structured Programming.....	48
How Do You Write Functions Like This?.....	49
Conclusion.....	49
Setup/Teardown/Includer.....	50
Bibliography.....	52
Chapter 4: Comments.....	53
Comments Do Not Make Up for Bad Code.....	55
Explain Yourself in Code.....	55
Good Comments.....	55
Legal Comments.....	55
Informative Comments.....	56
Explanation of Intent.....	56
Clarification.....	57
Warning of Consequences.....	58
TODO Comments.....	58
Amplification.....	59
Javadocs in Public APIs.....	59
Bad Comments.....	59
Mumbling.....	59
Redundant Comments.....	60
Misleading Comments.....	63
Mandated Comments.....	63
Journal Comments.....	63
Noise Comments.....	64
Scary Noise.....	66
Don't Use a Comment When You Can Use a.....	66
Function or a Variable.....	67
Position Markers.....	67
Closing Braces Comments.....	67
Attributions and Bylines.....	68

Contents	
Commented-Out Code.....	68
HTML Comments.....	69
Nonlocal Information.....	69
Too Much Information.....	70
Inobvious Connection.....	70
Function Headers.....	71
Javadocs in Nonpublic Code.....	71
Example.....	71
Bibliography.....	74
Chapter 5: Formatting.....	75
The Purpose of Formatting.....	76
Vertical Formatting.....	76
The Newspaper Metaphor.....	77
Vertical Openness Between Concepts.....	78
Vertical Density.....	79
Vertical Distance.....	80
Vertical Ordering.....	84
Horizontal Formatting.....	85
Horizontal Openness and Density.....	86
Horizontal Alignment.....	87
Indentation.....	88
Dummy Scopes.....	90
Team Rules.....	90
Uncle Bob's Formatting Rules.....	90
Chapter 6: Objects and Data Structures.....	93
Data Abstraction.....	93
Data/Object Anti-Symmetry.....	95
The Law of Demeter.....	97
Train Wrecks.....	98
Hybrids.....	99
Hiding Structure.....	100
Data Transfer Objects.....	100
Active Record.....	101
Conclusion.....	101
Bibliography.....	101

Clean Code

Contents	
Chapter 7: Error Handling.....	103
Use Exceptions Rather Than Return Codes.....	104
Write Your Try-Catch-Finally Statement First.....	105
Use Unchecked Exceptions.....	106
Provide Context with Exceptions.....	107
Define Exception Classes in Terms of a Caller's Needs.....	107
Define the Normal Flow.....	109
Don't Return Null.....	110
Don't Pass Null.....	111
Conclusion.....	112
Bibliography.....	112
Chapter 8: Boundaries.....	113
Using Third-Party Code.....	114
Exploring and Learning Boundaries.....	116
Learning log4j.....	116
Learning Tests Are Better Than Free.....	118
Using Code That Does Not Yet Exist.....	118
Clean Boundaries.....	120
Bibliography.....	120
Chapter 9: Unit Tests.....	121
The Three Laws of TDD.....	122
Keeping Tests Clean.....	123
Tests Enable the -ilities.....	124
Clean Tests.....	124
Domain-Specific Testing Language.....	127
A Dual Standard.....	127
One Assert per Test.....	130
Single Concept per Test.....	131
F.I.R.S.T.....	132
Conclusion.....	133
Bibliography.....	133
Chapter 10: Classes.....	135
Class Organization.....	136
Encapsulation.....	136

Contents	
Classes Should Be Small!.....	136
The Single Responsibility Principle.....	138
Cohesion.....	140
Maintaining Cohesion Results in Many Small Classes.....	141
Organizing for Change.....	147
Isolating from Change.....	149
Bibliography.....	151
Chapter 11: Systems.....	153
How Would You Build a City?.....	154
Separate Constructing a System from Using It.....	154
Separation of Main.....	155
Factories.....	155
Dependency Injection.....	157
Scaling Up.....	157
Cross-Cutting Concerns.....	160
Java Proxies.....	161
Pure Java AOP Frameworks.....	163
AspectJ Aspects.....	166
Test Drive the System Architecture.....	166
Optimize Decision Making.....	167
Use Standards Wisely, When They Add Demonstrable Value.....	168
Systems Need Domain-Specific Languages.....	168
Conclusion.....	169
Bibliography.....	169
Chapter 12: Emergence.....	171
Getting Clean via Emergent Design.....	171
Simple Design Rule 1: Runs All the Tests.....	172
Simple Design Rule 2-4: Refactoring.....	172
No Duplication.....	173
Expressive.....	175
Minimal Classes and Methods.....	176
Conclusion.....	176
Bibliography.....	176
Chapter 13: Concurrency.....	177
Why Concurrency?.....	178
Myths and Misconceptions.....	179

Contents	
Challenges.....	180
Concurrency Defense Principles.....	180
Single Responsibility Principle.....	181
Corollary: Limit the Scope of Data.....	181
Corollary: Use Copies of Data.....	181
Corollary: Threads Should Be as Independent as Possible.....	182
Know Your Library.....	182
Thread-Safe Collections.....	182
Know Your Execution Models.....	183
Producers-Consumers.....	184
Readers-Writers.....	184
Dining Philosophers.....	184
Beware Dependencies Between Synchronized Methods.....	185
Keep Synchronized Sections Small.....	185
Writing Correct Shut-Down Code Is Hard.....	186
Testing Threaded Code.....	186
Treat Spurious Failures as Candidate Threading Issues.....	187
Get Your Nonthreaded Code Working First.....	187
Make Your Threaded Code Pluggable.....	187
Make Your Threaded Code Tunable.....	187
Run with More Threads Than Processors.....	188
Run on Different Platforms.....	188
Instrument Your Code to Try and Force Failures.....	188
Hand-Coded.....	189
Automated.....	189
Conclusion.....	190
Bibliography.....	191
Chapter 14: Successive Refinement.....	193
Args Implementation.....	194
How Did I Do This?.....	200
Args: The Rough Draft.....	201
I Stopped.....	212
On Incrementalism.....	212
String Arguments.....	214
Conclusion.....	250

Contents	
Chapter 15: JUnit Internals.....	251
The JUnit Framework.....	252
Conclusion.....	265
Chapter 16: Refactoring SerialDate.....	267
First, Make It Work.....	268
Then Make It Right.....	270
Conclusion.....	284
Bibliography.....	284
Chapter 17: Smells and Heuristics.....	285
Comments.....	286
C1: Inappropriate Information.....	286
C2: Obsolete Comment.....	286
C3: Redundant Comment.....	286
C4: Poorly Written Comment.....	287
C5: Commented-Out Code.....	287
Environment.....	287
E1: Build Requires More Than One Step.....	287
E2: Tests Require More Than One Step.....	287
Functions.....	288
F1: Too Many Arguments.....	288
F2: Output Arguments.....	288
F3: Flag Arguments.....	288
F4: Dead Function.....	288
General.....	288
G1: Multiple Languages in One Source File.....	288
G2: Obvious Behavior Is Unimplemented.....	288
G3: Incorrect Behavior at the Boundaries.....	289
G4: Overridden Safeties.....	289
G5: Duplication.....	289
G6: Code at Wrong Level of Abstraction.....	290
G7: Base Classes Depending on Their Derivatives.....	291
G8: Too Much Information.....	291
G9: Dead Code.....	292
G10: Vertical Separation.....	292
G11: Inconsistency.....	292
G12: Clutter.....	293

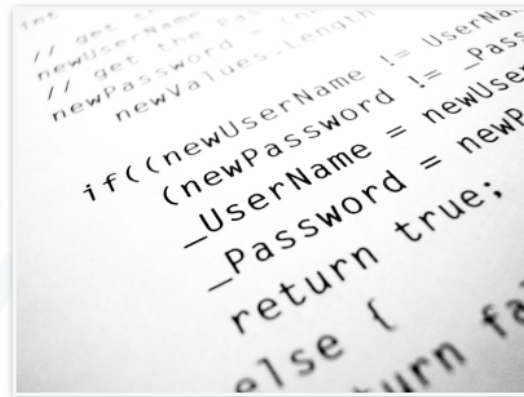
Contents	
G13: Artificial Coupling.....	293
G14: Feature Envy.....	293
G15: Selector Arguments.....	294
G16: Obscured Intent.....	295
G17: Mismatched Responsibility.....	295
G18: Inappropriate Static.....	296
G19: Use Explanatory Variables.....	296
G20: Function Names Should Say What They Do.....	297
G21: Understand the Algorithm.....	297
G22: Make Logical Dependencies Physical.....	298
G23: Prefer Polymorphism to If/Else or Switch/Case.....	299
G24: Follow Standard Conventions.....	299
G25: Replace Magic Numbers with Named Constants.....	300
G26: Be Precise.....	301
G27: Structure over Convention.....	301
G28: Encapsulate Conditionals.....	301
G29: Avoid Negative Conditionals.....	302
G30: Functions Should Do One Thing.....	302
G31: Hidden Temporal Couplings.....	302
G32: Don't Be Arbitrary.....	303
G33: Encapsulate Boundary Conditions.....	304
G34: Functions Should Descend Only.....	304
One Level of Abstraction.....	304
G35: Keep Configurable Data at High Levels.....	306
G36: Avoid Transitive Navigation.....	306
Java.....	307
J1: Avoid Long Import Lists by Using Wildcards.....	307
J2: Don't Inherit Constants.....	307
J3: Constants versus Enums.....	308
Names.....	309
N1: Choose Descriptive Names.....	309
N2: Choose Names at the Appropriate Level of Abstraction.....	311
N3: Use Standard Nomenclature Where Possible.....	311
N4: Unambiguous Names.....	312
N5: Use Long Names for Long Scopes.....	312
N6: Avoid Encodings.....	312
N7: Names Should Describe Side-Effects.....	313

Clean Code

How to know?

SOFTWARE
QUALITY
VISUALIZATION

Code Review



```
int  
// get the password  
newUserName = (newValues[0].Length  
// get the password  
newPassword = newValues[1].Length  
if((newUserName != UserNa  
(newPassword != _Pass  
_UserName = newUser  
_password = newP  
return true;  
else {  
    return fa
```

CR: What?

- ✓ Codes do right things?
- ✓ Codes do things right?

CR: Why?

The most inexpensive way to certify codes

CR: How?

- ✓ Look same code at same time
- ✓ Check logic and cleanliness

CR: Where?

Same space can see code together

CR: Who?

Every single team member

CR: When?

Everytime even a single line of code
added to the mainline

Not Easy to Follow

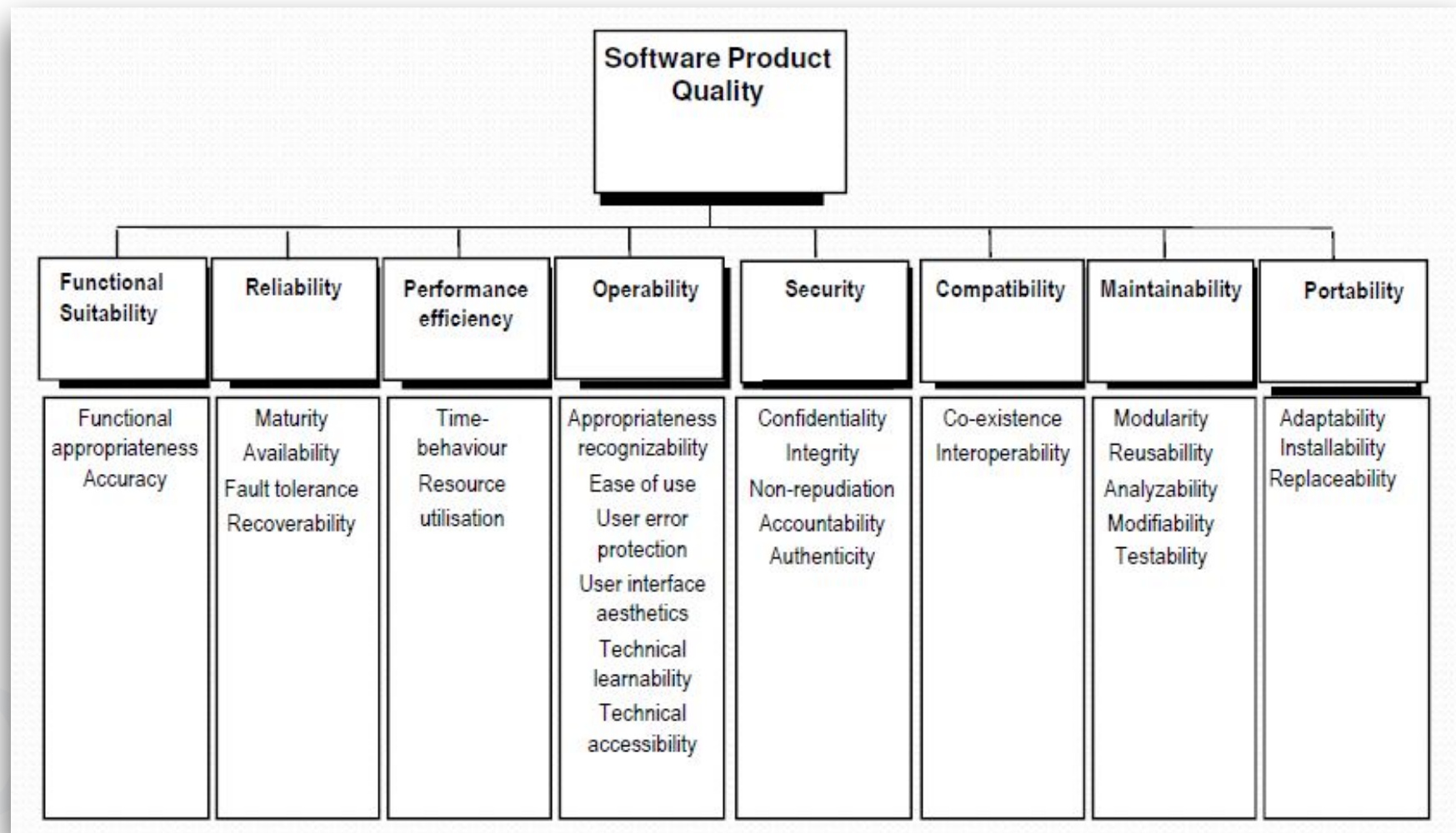
SOFTWARE
QUALITY
VISUALIZATION

Code Analysis

Static / Semantic / Dynamic

Static Analysis

Analyze syntax against rules
not executing software



Difficult

- ✓ Different tools for different languages
- ✓ Analyze which are enhanced
- ✓ Share with others

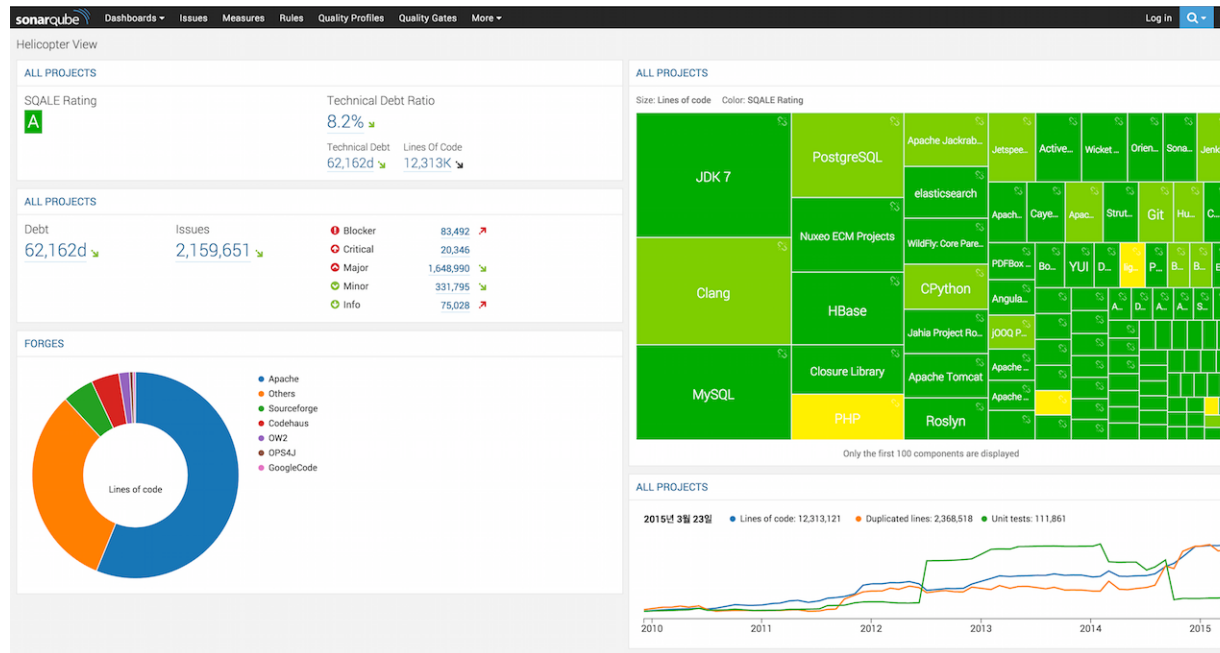


Finally on the stage

SOFTWARE
QUALITY
VISUALIZATION

5.1

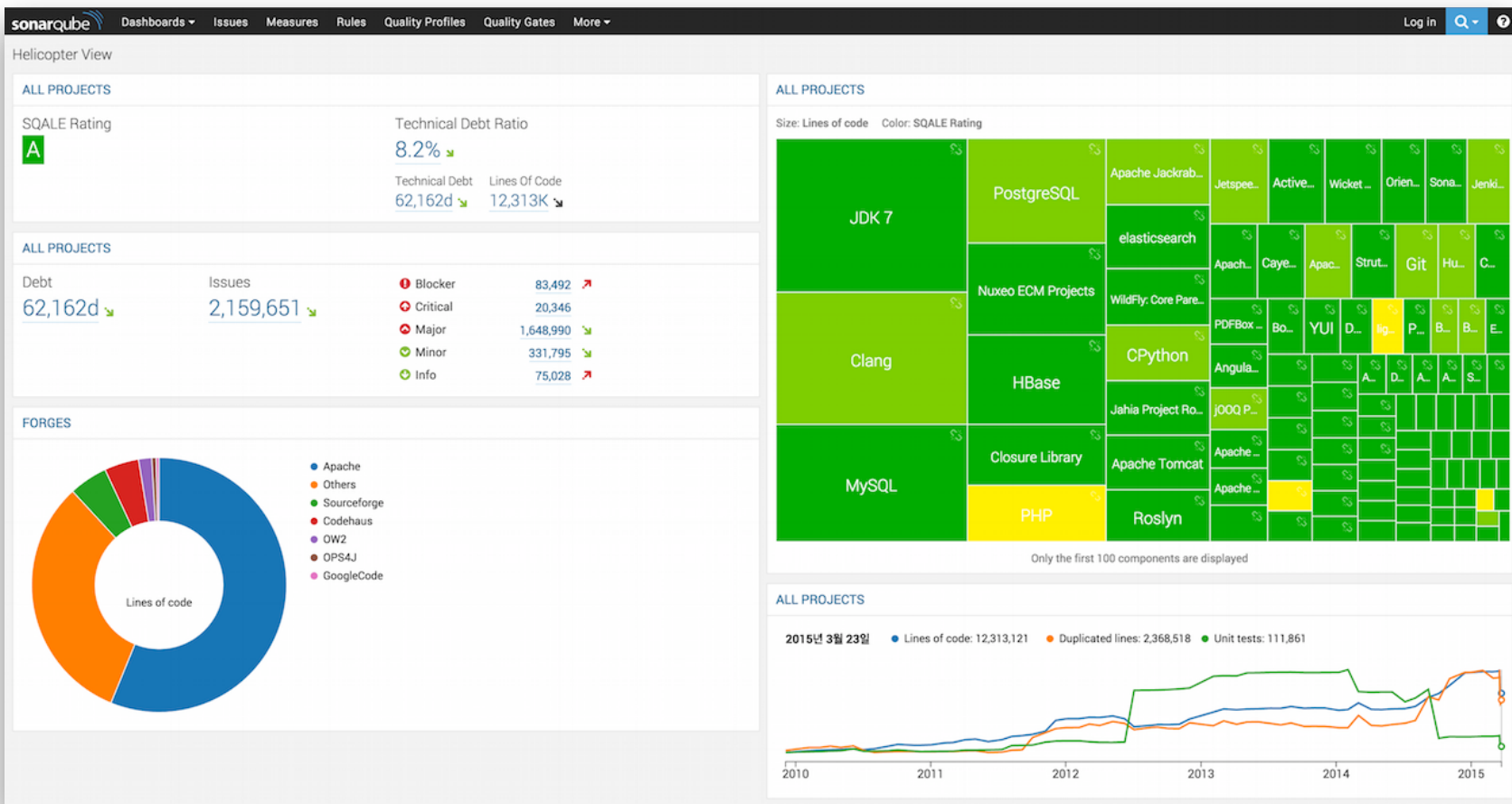
Lately released on April, 2015
Born in 2007



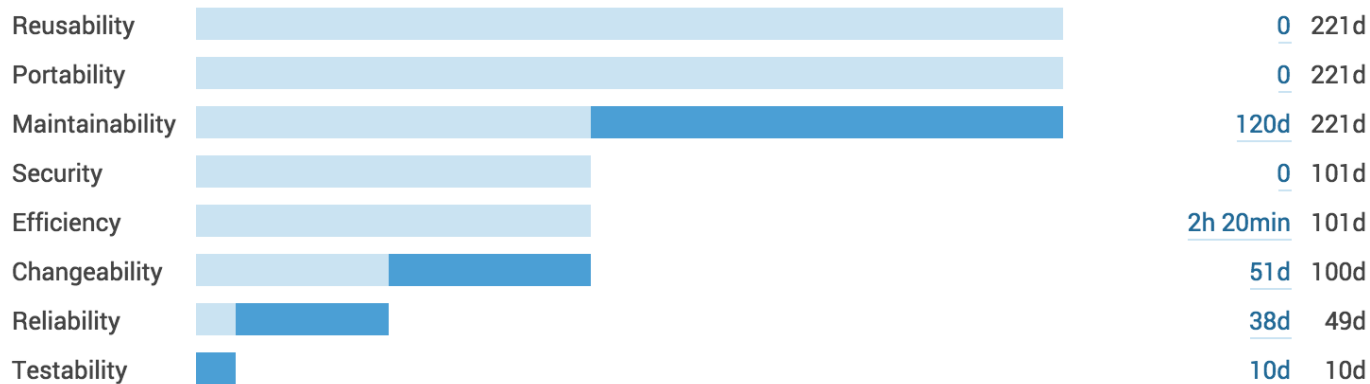
- ✓ Continuous inspection of code quality
- ✓ Fully-integrated web platform
- ✓ Easy integration to environment
- ✓ + FREE

CI Continuous Inspection

- ✓ Past/ Present/ Prediction/ Publication
- ✓ Technical debt
- ✓ Historical SQ visualization

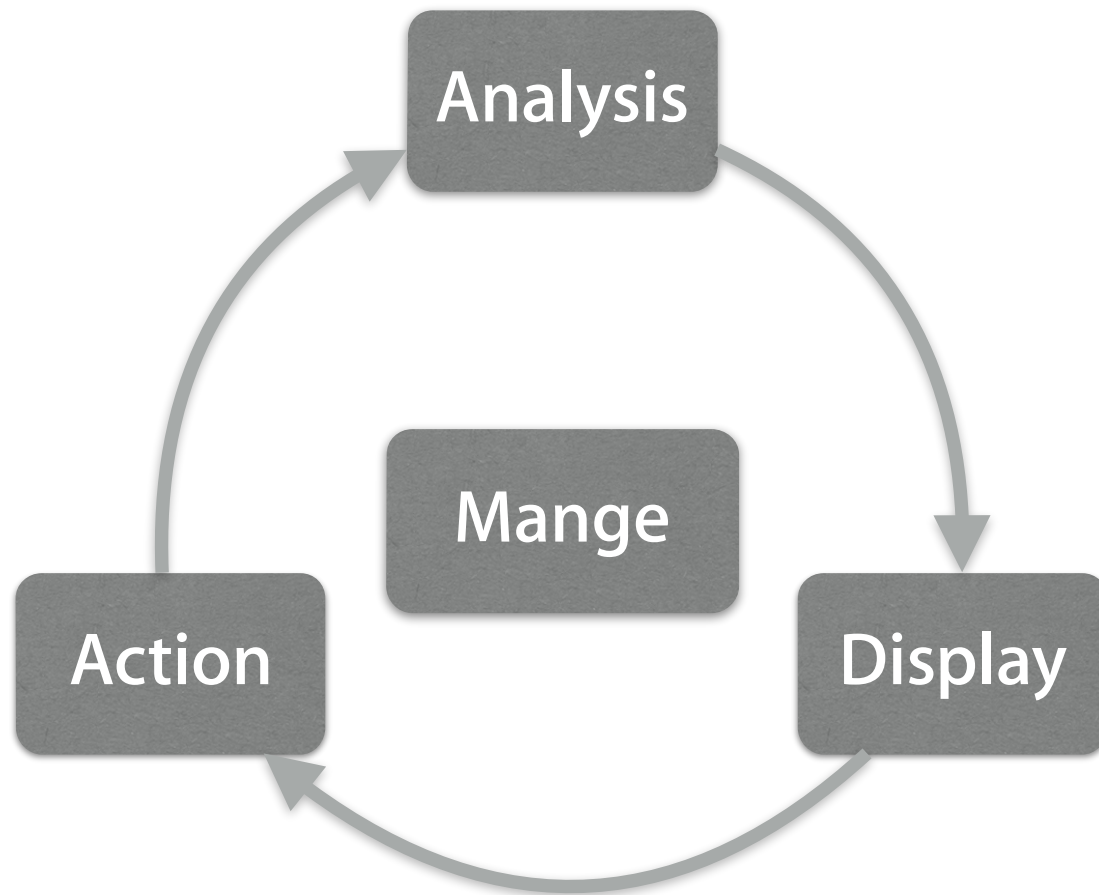


Technical Debt Pyramid



Fully-integrated

- ✓ Complete workflow
- ✓ Single web platform
- ✓ Various languages & rules



Easy Integration

- ✓ Continuous Integration^{CI}
- ✓ Widely used IDEs
- ✓ Various SCMs



LDAP
Lightweight Directory
Access Protocol

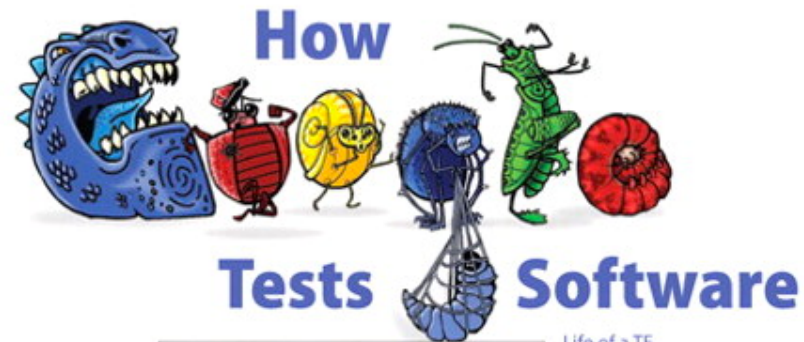


NetBeans



Matters to Testing

Clean code is also important for testing



Help me test like Google

Life of a TE
Life of an SET
Interviews with Googlers
and more

SWE

SET

TE

STE

Software Engineer in Test

- ✓ Focus on testability, test infra-structure
- ✓ Enhance quiality and test-coverage
- ✓ Make feature relate to quality

For Testers

- ✓ API documentation
- ✓ Module dependencies
- ✓ Test coverage

For Testers

- ✓ To make better quality software early
- ✓ To detect defects more early
- ✓ To do other than noarmal BBT

Demonstrations

SOFTWARE

QUALITY

VISUALIZATION

Summary

- ✓ Clean codes = Culture & Habit
- ✓ SQ+CI strongly support to enhance SQ
- ✓ Try it today :-)

REFERENCES

.Clean Code

A Handbook of Agile Software Craftmanship
Written by Robert C. Martin

.How Google Tests Software

Help to test like google. Life of TE, Life of STE
Interviews with Googlers and more
Written by James Whittaker, Joson Arbon & Jeff Carollo

.SonarQube User Guide

<http://www.sonarqube.org/documentations>

Q&A

✉ creatinov.kim@gmail.com

💻 www.creatinov.org

📘 <https://www.facebook.com/groups/korea.sonarqube.user.group/>

Thank you

공개SW가 테스트를 만나다!
14⁴⁵-15³⁰/20150429/수
강남SC컨벤션12F아나이스홀