How to improve the quality of your application

(I wish I'd known this earlier!)

Ioannis Kolaxis – Coding Architect / Senior Expert

Tuesday 10th December 2019 Java2Days, Sofia / Bulgaria

Trusted partner for your Digital Journey



Our application: CMP

 Automates the configuration & provisioning of our products, achieving significant time savings for our service.

	UNIFY	Common	Mai	nagement Pl	atform 🛛	omain: system			User: administ	trator@system Sett	ings Help Logout
Ē	Configuration	Maintenance		User Managemen	t 🛛 🗌 Fault Mar	agement	Performance	Manageme	ent Accou	ınting 🔰 4 📕	138 📕 17 📕
	Administra	tion	ا 🌮	Jsers & Resource	s						?
2	Users & Resources	5	1 U	Jse the Column Settings Isers and Resources dis	option to select the r	number and order (of columns to be disp to download the list	olayed. Use Se +	arch or Advanced	Search to narrow do	wn the number of
2	Localities										
പ്പ	User Lemplates		Searc	h for: in	Last Name	✓ in All Locali	ties 🗸 with An	y Resource	Search	Show all	Advanced
	Provision	ing							More V	Add Edit	Delete
830	CSV Import		Sel:0	Items/Page: 100 🗸	🔲 All:6 🏭						
	Users from HPUM LDAP			Last name 🔺	First name	Bus.	Phone 1	Resources	User Template	Status	
*	Users from UC Acc	ounts		🚱 Desarti	Athina	+30 (210) 8189-613	00	osvUserTemplate	•	•
2	Users from OSV Su	bscribers		Fotopoulos	Dimitrios	+30 (210) 8189-140	86		•	•
				💱 Karakatselos	Konstantinos	+30 (210) 8189-847	00	osvUserTemplate	•	•
				💱 Kolaxis	Ioannis	+302	108189858	00	osvUserTemplate	•	•
				Nanouris	Ioannis	+498	970071230	0		•	•
				Pegiou	Vasiliki	+30 (210) 8189-793	00	osvUserTemplate	٢	•



Software quality issues



Are you working for a software product, where ...?

- Customers keep complaining about bugs
- New features take too much

time to be implemented



What can you do?

- Can you improve the quality of your software?
- How?





Customer tickets ✓

• We usually measure *quality* via customer tickets:







Code coverage ✓

When we refer to *quality*, we usually think of code coverage!

🎡 Jenkins						🔍 search	Iog in sign up
Jenkins > CodeCoverage > Code_Cov	verage_Report > UC V9.3.8.13						
 Back to Project Status Changes Console Output View Build Information Coverage Report HTML Report Previous Build 	Code Coverage Cobertura Coverage Re Trend	eport			Increa • fror • to 7	ased cove n 67% (Fe 72% (Jun 3	rage: eb 2017) 2018)
	Project Coverage summary	_					
	Name Cobertura Coverage Report	Packages 96% 208/216	90% 1710/1905	Classes 86% 2177/2519	Methods 67% 23073/34331	Lines 72% 199797/278018	57% 108453/190369



Should you pay off your debt?





Old code is more reliable



Do <u>not</u> touch old code! You will probably introduce new defects!

"If a module is, on the average, a year older than an otherwise similar module, the older module will have roughly a third fewer

faults."

7

T. L. Graves, A. F. Karr, J. S. Marron and H. Siy, "Predicting fault incidence using software change history" in *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653-661, Jul 2000.



Stop creating new debt

- Install SonarLint plugin in your IDE.
- It helps you detect, and fix quality issues as you write code.
- Download at:

www.sonarlint.org

Edit Source	Refactor Navigate Search Project Run Window Help		
- 🛛 🕼 ≙	! 🖳 ! 🔌 💷 📾 🖉 32. ⊙() 🔜 🕱 ! 🎋 ▾ 🔾 ▾ 🏰 🥝 ▾ 🤔 🗀 🖋 ▾ 🖓 ♥ 💋 🐲 🚇 🗐 🔳 ! 😓	• 🖓 • 🍤 🎸	þ ▼ ⇔ ▼
		Quick Access	R 🛛 🌣
(a			
UserMan	agementImpl.java 💥 🕎 RetrieveUserTimpltForUsersDataControlImpl.java		0 % - 0
6300	<pre>writeCreationDataToFile(reportFile, resultsLog.printLog()); </pre>	^ -	🗊 🖻 🔤 🖁
6302	BUIKEAITUMUSERSKEPIY FEPIY =new BUIKEAITUMUSERSKEPIY(); renlv.setNumberOfSuccessfulfdited(lsers(new Integer(successfullvEdited));		\bigtriangledown
6303	reply.setNumberOfUnSuccessfulEditedUsers(new Integer(unsuccessfullyEdited));		j= ^
6304	reply.setBulkEditCanceled(canceledReply);		i [®]
6305	LOG.into("Bulk edit of UM users ended");		i=
6307	regContext.reply(reply);		• •
6308	} catch (Exception e) {	8	P
6309	UserManagementException ex = new UserManagemel Press 'F2' for focus	-	·
<u> </u>			
Problem	; 🐵 Javadoc 🗓 Declaration 🛷 Search 🖞 History 📃 Console 🐐 Debug 😁 SonarLint Rule Description 🛿 뗾 Git Repositories	5	State
Constant			-
Construc	tors should not be used to instantiate. String, and primitive-wrapper classes (squid:S2129)		
🛞 Code s	nell 🔕 Major		
Constructo	re for Facility and the objects used to user primitives should never be used. Doing so is less clear and user more memory th	an cimply using	the
desired va	to for schings, and the objects used to wrap primitives should never be used. Doing so is less clear and uses more memory the	an simply using	uie
uesireu va	de in the case of strings, and using valueor for everything else.		=
Further, th	ese constructors are deprecated in Java 9, which is an indication that they will eventually be removed from the language altoget	her.	
Noncom	pliant Code Example		
String e	<pre>mpty = new String(); // Noncompliant; yields essentially "", so just use that.</pre>		
String	onempty = new String("Hello world"); // Noncompliant		
Double r	<pre>yUouDle = new UouDle(1.1); // Noncompliant; use valueUt interes = new Tetran(1); // Noncompliant;</pre>		
Boolean	Integer - new Integer(I); // Noncompliant		
DOOTGUI	boor - new boorcan(cracy, // noncomprishe		



Stop creating new debt

Setup Quality Gates in SonarQube

sonarqube Projects	Issues Ri	iles Q	Quality Profiles	Quality Gates	Administration		C	λ Search fe	or projects, si	ub-projects	and files	
Quality Gates	Crea	te	SonarQube	way					Rename	Сору	Set as Default	Delete
SonarQube way			Conditions Only project r	measures are cheo	cked against thresholds. Sub-project	s, directories and files	are ignored. <u>More</u>					
			Metric			Over Leak Period	Operator		Warning	Error		
			New Blocke	r Issues		Always	is greater than	T		0	Update	Delete
			New Bugs			Always	is greater than	*		0	Update	Delete
			New Critical	Issues		Always	is greater than	-		0	Update	Delete
			New Major	Issues		Always	is greater than	*		0	Update	Delete
			New Vulner	abilities		Always	is greater than	Ŧ		0	Update	Delete





- As a *developer*, where do you spend most of your time?
 - A. Reading existing code,
 - B. Writing new code,
 - C. Waiting for a full build to complete,
 - D. Other





• As a *developer*, where do you spend most of your time?

A. Reading existing code,

- B. Writing new code,
- C. Waiting for a full build to complete,
- D. Other



Just think ...



Which parts of your code do you read most often?



Data never lies

Use *git* to find out where you spend most of your development efforts:

git log --format=format: --name-only | egrep -v '^\$' | sort | uniq -c | sort -r >

files_change_frequency.txt

258 usermanagementportlet/.../UserManagement_de.properties

```
per file 250 usermanagementportlet/.../UserManagement_en.properties
```

```
227 usermanagement/.../RetrieveUserTmpltForUsersDataControlImpl.java
```

- 205 usermanagement/.../UserManagementImpl.java
- 154 usermanagement/.../EditUserResourceTemplateRulesBean.java
- 135 usermanagementportlet/.../AddEditUserBean.java
- 109 usermanagementportlet/.../ConfigureNewUserResourceBean.java
- 103 usermanagementportlet/.../addEditUser.jsp



Commits

The pattern

- Only a few files change frequently!
- This is where you spend most of your time!

From a total of 10.007 files:

- 11 files \rightarrow more than 100 commits
- 91 files \rightarrow 31 < commits < 100
- 455 files \rightarrow 10 < commits < 30
- 9.450 files \rightarrow *less than 10 commits*





Refactor frequently-changing files



A well-aimed refactoring will help you:

- Spend less time to read code
 & extend functionality.
 - Become more productive!



Changing files predict system failures

- "Churn measures based on counts of lines added, deleted, and modified are very effective for fault prediction." R. M. Bell, T. J. Ostrand, and E.J. Weyuker, "Does Measuring Code Change Improve Fault Prediction?", ACM Press, 2011.
- Files involved in a lot of bug fixing activities are most likely to be defective

R. Moser, W. Pedrycz, and G. Succi, "A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction", Proceedings of the 30th International Conference on Software Engineering, 181-190, 2008.



Focus your Quality Assurance efforts

- Do not waste your time testing *mature* functionality (=components that do not change).
- Focus all your testing efforts on the frequently-changing parts; those are most likely to fail!





Ask the right questions



What is the coverage of your new/changing code?





Identify stable components

- Files not changed in the past years → stable components → mature features
- Is every **mature feature** still used by your customers?
 - If a feature is **not** used, then **delete** its code!
 - Else, extract stable features in separate libraries.



Go faster with deleted/extracted code

- Save time from your builds.
- Achieve faster onboarding of new developers, by:



- Focusing only on actively developed code.
- Not having to familiarize with old/stable code.



Measure code complexity



 Gain more insight, by measuring code complexity for each one of the frequently changing files.

- Language-neutral metrics for code complexity:
 - Number of lines
 - Number of tabs



Tabs increase complexity



How many times did you provide a bug fix, by adding a nested conditional in your code? if (...) { for (...) { \longrightarrow if (customerSpecificSetup) { tabs // Do some magic, so that the // application works for this customer!



Rising complexity calls for refactoring

RetrieveUserTmpltForUsersDataControlImpl.java







Our #1 priority for refactoring





Our #1 priority for refactoring







Refactor frequently changing files

- The identified files are being changed by many developers in parallel.
- Is it feasible to perform refactoring on a private branch?
- Can we afford to stop development, while someone works for a *long time* on refactoring the identified files?



Break large file by responsibilities





Divide and conquer



• When you refactor,

always try to stabilize new/changing code!



Stabilizing code by refactoring





Do you remember Windows Vista?

- Released on 8th November 2006.
 - > 50 million lines of code.
 - ~ 2.000 developers.





Organizational structure vs Quality

 Microsoft measured several organizational metrics, and studied their correlation with the defects of Windows Vista.

Organizational metric	Assertion
Number of Engineers	The more people who touch the code, the lower the quality.
Number of Ex-Engineers	A large loss of team members affects the knowledge retention, and thus quality.
Organization Intersection Factor	The more diffused the different organizations contributing code, the lower is the quality.

• Can the **structure** of **your organization** affect the **quality** of your

software application?

N. Nagappan, B. Murphy, and V.R. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study", ACM, 2008.



Organizational structure impacts Quality

 Organizational metrics are better predictors of failure-proneness than the traditional metrics used so far, such as code coverage, code complexity, etc.

Model	Precision
Organizational structure	86,2%
Code coverage	83,8%
Code complexity	79,3%
Code churn	78,6%
Dependencies	74,4%
Pre-release bugs	73,8%

N. Nagappan, B. Murphy, and V.R. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Case Study", ACM, 2008.



More organizational metrics

- In another research, focused on Windows 7, Microsoft distinguished between the following kinds of developers, Windows 7
 depending on their commits for a given component:
 - Owner: has the most commits to that component.
 - **Major contributor:** has *more than 5%* of total commits.
 - **Minor contributor:** has *less than 5%* of total commits.



<u>C.Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't Touch My Code! Examining the Effects of Code Ownership on</u> <u>Software Quality", ACM, 2011.</u>

Effects of minor contributors

- The researchers concluded that:
 - "The number of **minor contributors** has a strong positive relationship with both pre- and post-release failures ..."
 - "Higher levels of **ownership** for the top contributor to a component results in fewer failures when controlling for the same metrics, but the effect is smaller than the number of minor contributors"

C.Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't Touch My Code! Examining the Effects of Code Ownership on Software Quality", ACM, 2011.



Gain insight into your components

• In one of our software components, we had a total of 427 commits:

Commits per developer

 The top contributing Value Metric 20.37% developer made 87 Minor 15 3.98% contributors commits: 4.22% Major 6 87/427 = **20,37%** ownership contributors 4.92% 14.52% Total 21 4.92% contributors 5.85% **Ownership** 20,37% 10.30% 7.96% 8.20%



Gain insight into your components



- In another software component, we had a total of 253 commits for the same period:
 - The top contributing developer made 73 commits:



Commits per developer

Know where you are standing ...

Metric	Component A	Component B
Minor contributors	15	3
Major contributors	6	6
Total contributors	21	9
Ownership	20,37%	28,85%

- Which component will probably have more defects?
- Where would you focus your testing efforts?



Beware of minor contributors!

Metric	Component A	Component B
Minor contributors	15	3
Major contributors	6	6
Total contributors	21	9
Ownership	20,37%	28,85%

- More minor contributors \rightarrow More defects
- Bigger ownership
 → Less defects



Use metrics to build better software



- Minor contributors must be **consulting** a major contributor of a component <u>before</u> making any changes to it.
- Pay more attention when **reviewing**
- code submitted by minor contributors.
- More extensive testing should be
 - performed for components with *low*

ownership.



Planning new features

- A customer asks for a new feature to be implemented, but the major contributors of that component are <u>not</u> <u>available</u>. What will you do?
 - Ask from minor contributors, to start implementing this new feature right away, or
 - Delay the implementation of the feature, until one or more major contributors are available?



Learn your contributors

• Use *git* to find out all the contributors for a component:



 Or, to limit the results to contributors after a given date (e.g. due to an organizational restructuring) git shortlog -s --after=2018-05-01 your_component > contributors.txt



Summary of proposed actions



- 1. Stop creating new quality issues.
- 2. Don't touch old code.
- 3. Refactor your most complex, frequently changing files.
 - Focus your testing on frequently changing files.
- 5. Pay attention to minor contributors.



How do you build quality software?



Let's share our knowledge & experience!



Thank you!

Email : ioannis.kolaxis@atos.net **Twitter** : @IoannisKolaxis

Atos, the Atos logo, Atos Codex, Atos Consulting, Atos Worldgrid, Worldline, BlueKiwi, Bull, Canopy the Open Cloud Company, Unify, Yunano, Zero Email, Zero Email Certified and The Zero Email Company are registered trademarks of the Atos group. June 2016. © 2016 Atos. Confidential information owned by Atos, to be used by the recipient only. This document, or any part of it, may not be reproduced, copied, circulated and/or distributed nor quoted without prior written approval from Atos.

