How to Implement Zero-Debt Continuous Inspection in an Agile Manner

A Case Study

Brian Chaplin
Brian.chaplin@gmail.com
Bchaplin1 on twitter

Business Context

- Large project, semi-Agile
- Corporate Six Sigma program, not IT
- IT history of OO pair programming
- Management goal of 80% test coverage
- Continuous integration build/deploy
- Productive open source environment
- Velocity always trumps quality

Characteristics of 2 Case Studies

Category	Java Open Source	C# .Net
Developers	100	175
NCLOC (non-comments)	868,000	784,000
Commits per day	25	36
Classes	12,700	12,600
Unit tests, Coverage	37,000, 82%	22,000, 63%
File changes per day	225	500
Technical debt per day	10	50
Code quality since	2009	Dec. 2012

Architecture Requirements for Code Quality (CQ) Statistics

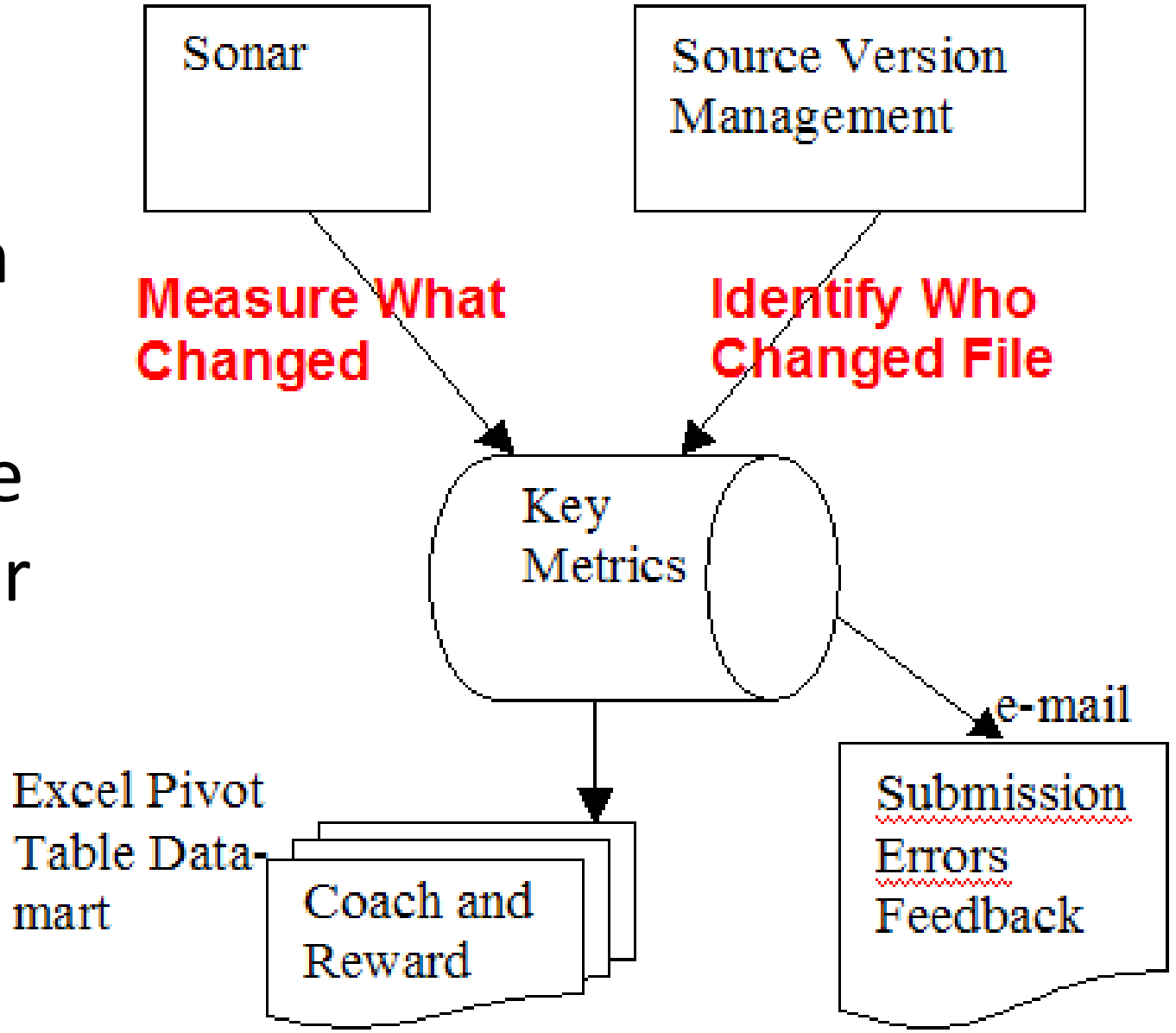
- Must come from existing continuous integration process
- Available within 24 hours
- Meaningful to developers, leads and management
- Must involve reviewers
- Must tie to accountability points
  - Leads, reviewers, submitters, project, business function

Architecture Overview

- Extract/Transform/Load (ETL)
- Input
  - Source code management system (SCM)
  - Project management system
  - Code quality metrics
- Output
  - SQL database
  - Excel pivot table reports
  - Defect tickets (project management system)
  - Email
  - Ad hoc management reports
  - Program scores and developer contribution

Architecture ETL Overview

- Extract, transform and load
- Match the committer with the quality change



Extract

- Sonar REST API
- SCM API
  - Perforce
  - TFS
- Project management system API
  - JIRA
- Match
  - SQL join
  - Pattern match Sonar artifact to TFS file path name

Transform

1. Save key quality stats
2. Determine time span between Sonar runs
3. Save the change lists for that time span
4. Save the before and after Sonar data for each change in the change lists

Load

- Persist via Spring JDBC
  - To Oracle or SQL Server
  - 25 tables in the schema
- Before and after quality stats
  - By file (fully qualified class name)
  - Also by package for Java
- Committer and reviewer
  - Email
  - Name
- Commit ticket info
  - Who authorized the ticket

Establish the Quality Database

- Leverage normalized schema via SQL
  - 280 SQL views encapsulate table joins
- Guard against corrupted quality data
  - Test or build failures
  - Sonar may record lower stats
  - Rollback a bad build
- Design for mash-ups
- Combine code quality with runtime defect reporting

Maximum Report Flexibility

- Maximize flexibility to enable timely and targeted reporting
- Excel pivot table as data-marts
- Excel output via Apache POI Java library
- Refreshed Excel reports accessed via Email hyperlinks or served up by web server

Agile Email

- Prompt technical debt notification
  - 3 times a day or after every CI build
  - Directly to committer and the reviewer
  - After each continuous inspection build
  - Direct Sonar hyperlink to the degraded file
- Daily contribution
  - Best contributors summary
  - Personalized contribution detail to each committer
- Links to full reports, metrics detail, wiki

Sample Violation Email

Submission Warning(s)

You are receiving this message because even though the code quality may have been enhanced, the submission(s) below have decreased the code quality (coverage, compliance, and/or uncovered complexity) or they didn't meet the standards for a new class. Please review these submissions with your reviewer and take the appropriate action.

Project ID	Change ID	Class	Uncovered Lines/Branches	Static Code Analysis Debt	Excessive Class Complexity	Excessive Function Complexity
ticket1	580932 4/18/2013	Class1	9	9		
ticket1	580932 4/18/2013	Class2	9	0		
ticket1	580902 4/18/2013	Class3	4	0	14	2.8
ticket1	580902 4/18/2013	Class4	0	0	26	5.2
ticket1	580902 4/18/2013	Class5	23	0	13	2.6

Best Practices

- Staff a code quality desk
  - Knowledge clearinghouse
- Use code reviewers
- Stabilize build and project structure
- Establish static code analysis rules and rarely change them
- Developers must be able to clear unfixable debt
- Run the ETL at least daily
- Keep the database accurate
- Track both contribution and debt
- Recognize code quality champions
- Use uncovered line/branch count not percentage

Technical Debt Defined

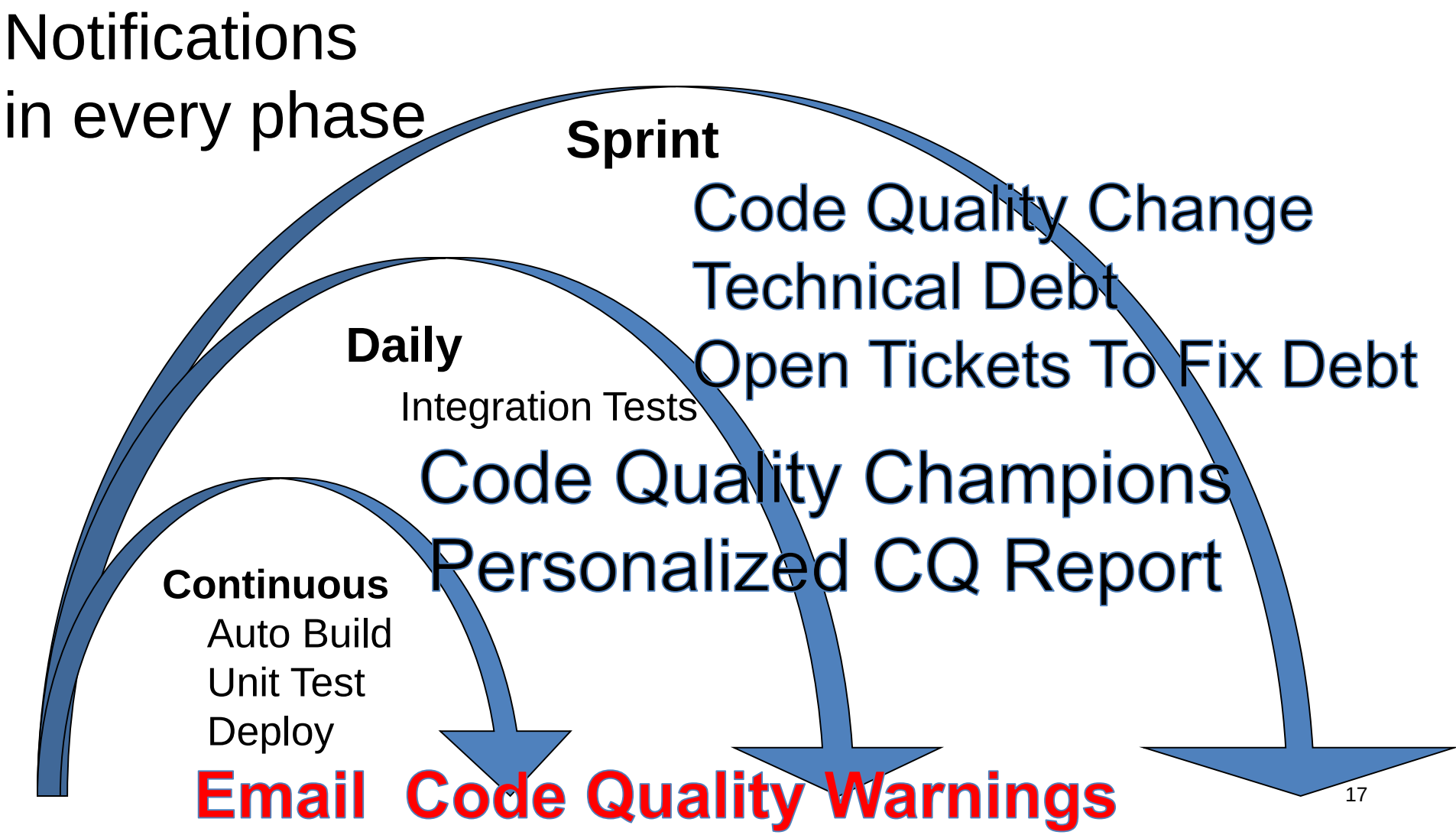
Coverage	Uncovered conditions Uncovered lines
Complexity	Average method complexity Total class complexity
Compliance	140 static code analysis rules Critical, major, minor weights
Comments	Comment density
Duplication	Duplicated lines
Organization	Circular dependencies

Tech Debt Reported

Sprint Debt				Metric							
				static violations			coverage		too complex		duplicated
lead	reviewer	devr	class	critical	major	minor	branch	line	method	class	lines
lead1	reviewer1	devr1	class1		-1		-60	-127			
			class2			-2	-4	-20			
			class3				-5	-11			
			class4		-1		-4	-9			
			class5				-2	-11			
			class6	-3			-4	-1			
			class7					-8			
			class8					-8			
			class9					-8			
			class10		-3						
			class11	-2	-1						
			class12			-2					
			class13					-1			
			class14			-1					
			class15		-1						
		devr2	class6		-1			-1			
			class16		-1						
	reviewer2	devr3	class17			-1		-16			
			class18					-2			
		devr4	class19		-4		-3	-5			
			class20				-3				

- Metric
- Submitter
- Lead
- Reviewer
- Class
- Sprint
- Date

Agile Code Quality Baked In



Automating Continuous Inspection Reporting

- Necessary for large projects
  - Agile and timely reporting within hours
- Necessary for zero-debt
  - Architects only have time for the big issues
  - Computer handles the smaller quality defects

Allow for Exclusions

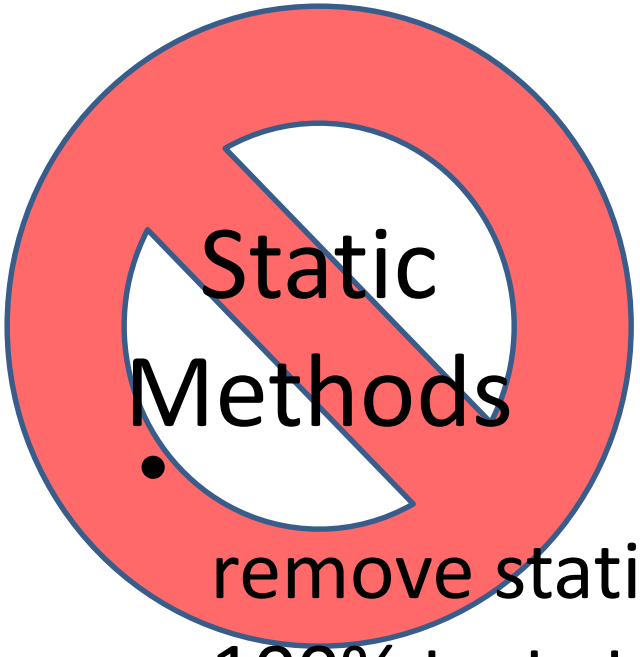
- Integrations, class rename, code moves
- Classes exempted from quality
  - Registries
  - Test support
- Unfixable debt
  - Caused by someone else
  - Unreachable test cases
  - Exceptions to the metrics standard
- 0.5% are excluded

80/20 Rule Vs. 100% Coverage

- Which 20% is uncovered?
  - null testing
  - value object setter/getters
- Integration vs. unit tests
- How do you automate the 20% exclusion?
- Test harnesses
  - Value objects
  - SOAP services
  - Workflows

Lessons Learned

- Communicate the benefits of code quality
  - Maintainability
  - Less runtime defects, explain carefully
  - Working with a net
- Help new developers with un-testable code
  - Train and mentor
  - Consider refactoring to testable code first, then write the test
- Make sure management understands personnel impact



Write Testable Code

- 1,000 class re-factor required to remove static methods before implementing 100% test standard
- Google has good guidelines on writing testable code
- Follow Law of Demeter design guideline
- Constructor injection
- Consider functional programming library

Manual Adjustments and False Positives

- Generated and third-party code
- Branch is dead, fixed in main
- Same class submitted by 2 different submitters in the same build cycle
- Un-testable until it's re-architected
- It wasn't my change that caused it, spurious test coverage change
- Code will be deleted or re-factored soon
- Static Code analysis rule changed during the build
- Grandfathered because class was just renamed or moved, no code modified
- Reviewer allows selective suppression of static code analysis violations via source code annotation and explanatory comment

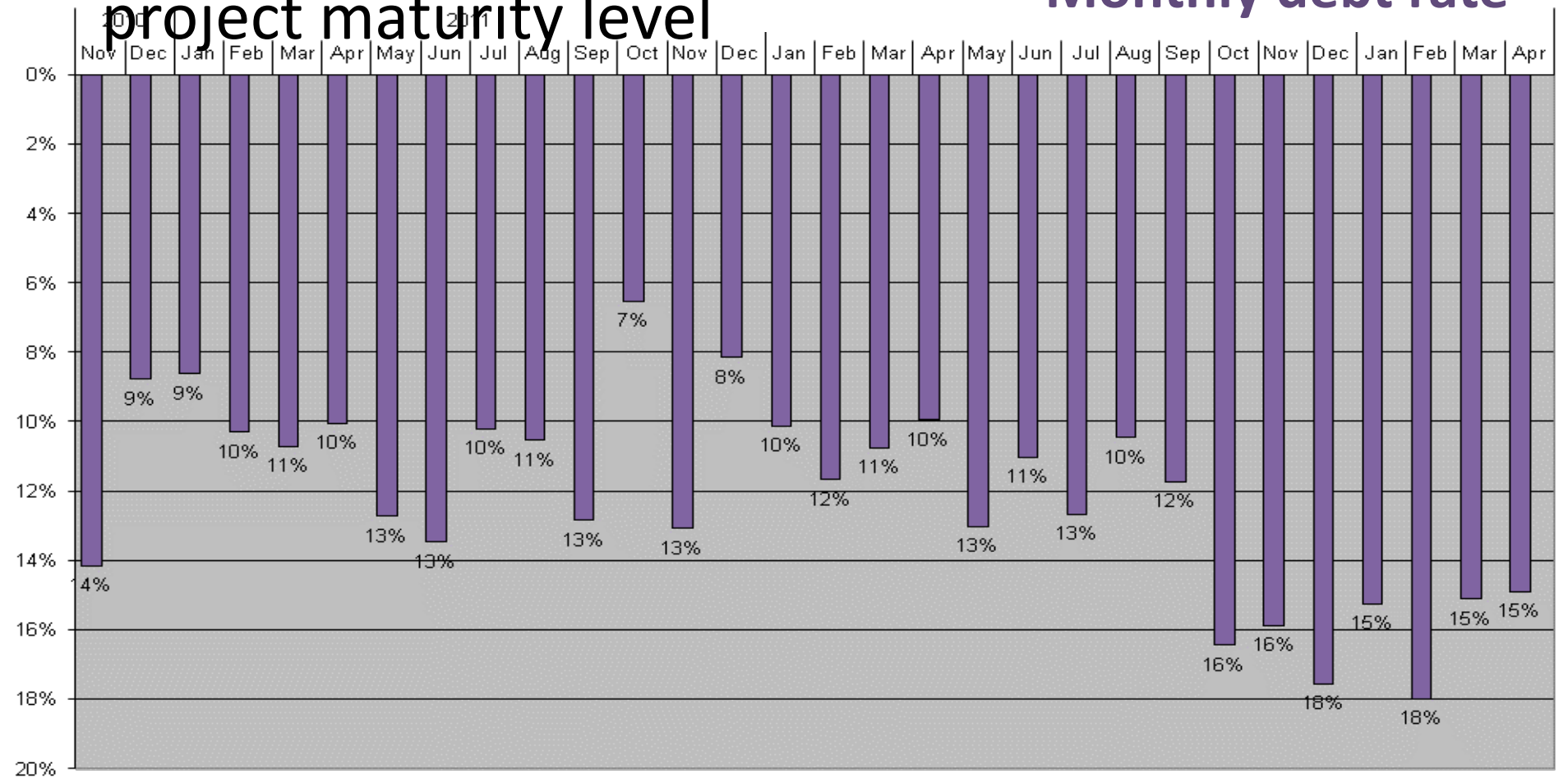
Reviewers Guard Against Doing the Wrong Thing

- Meaningless comments
- Unit tests just for coverage that assert nothing
- Allow duplicated code exception for some value objects
- Allow some large case statements
- Disallow breaking a branch evaluation into meaningless boolean evaluation operator

Managing the Debt

- Debt rate is about 15% for both projects
- Varies widely by developer
- Depends on management and project maturity level

Monthly debt rate



How to Manage the Debt

- Immediate feedback
- Open defect tickets at key points
- Train
- Assist developers one on one
- Encourage re-factoring to write testable code
- Code a little, test a little
- Keep management aware
- Continually monitor, don’t let it get out of hand
- Recognize the champions

Measurement Side-effects

- Technical debt metrics are informational
- Quality contribution metrics are motivational
- Quality desk must ensure that the right thing is done
- Can all programmers improve their code?
- How to safely re-factor un-testable code so all can contribute

Count	282	8,447
Average value	**NPE**	**Not**
Function complexity	2.90	1.88
Lcom4	1.98	1.14
Weighted violations	7.52	2.70
Complexity	**66**	**11**
Coverage	85%	83%
Duplicated lines	1.99	5.61
Comment lines density	28%	32%
Statements	**144**	**25**
Score	1.53	1.73
Fan-out	9.45	3.16

What Code Causes NPEs?

Probability =exp(a + b*WEIGHTED_VIOLATIONS + c*COMPLEXITY -d*COVERAGE + e*COMMENT_LINES_DENSITY - f*SCORE)

11/20 top NPEs predicted

probability	class name	CXTY	SCORE
100%	class1	528	1.13
100%	class2	523	1.12
99%	class3	523	1.45
99%	class4	523	1.25
99%	**class5**	**464**	**0.89**
86%	**class6**	**428**	**1.10**
90%	**class7**	**416**	**1.12**
95%	**class8**	**399**	**0.95**
95%	class9	385	0.83
92%	class10	372	1.24
98%	**class11**	**369**	**1.07**
96%	**class12**	**345**	**1.18**
96%	class13	341	1.33
88%	**class14**	**328**	**1.02**
87%	**class15**	**318**	**1.09**
85%	class16	316	0.89
77%	**class17**	**310**	**1.11**
75%	**class18**	**309**	**1.24**
93%	**class19**	**305**	**1.05**
91%	class20	279	1.32

Code Quality Take-away Messages

Quality Motivates and Enhances Productivity

- Tom DeMarco & Timothy Lister, *Peopleware:*
- “Quality, as defined by the builder (far beyond that required by the end user), is a means to higher productivity”
- “Quality is free, but only to those who are willing to pay heavily for it”
- Why management will pay for a code quality system

Implement CQ Continuous Inspection

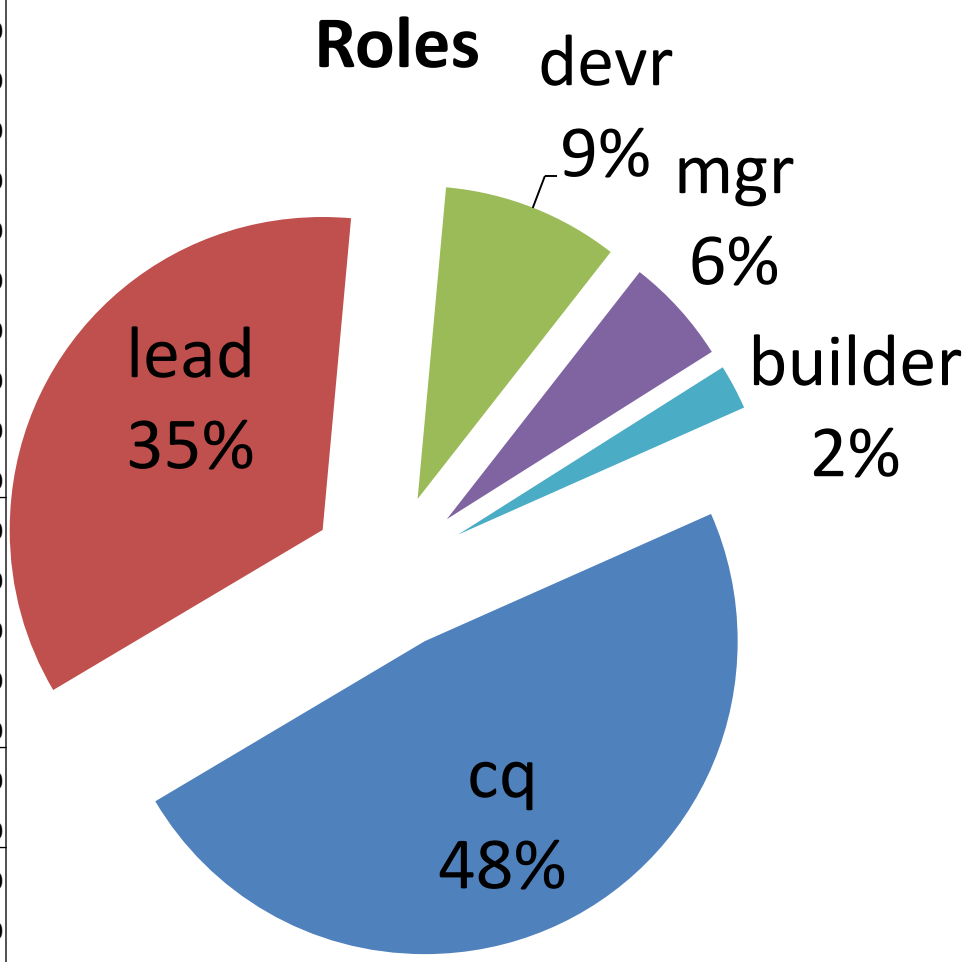
- Establish CQ point of accountability
- Promote CQ
- Track technical debt
  - Require remediation
- Scales to large projects
  - Multi-branch
  - Multi-language

- Targeted reporting
  - End of Sprint or production release branch
  - By lead, committer, reviewer
  - Commit quality trend

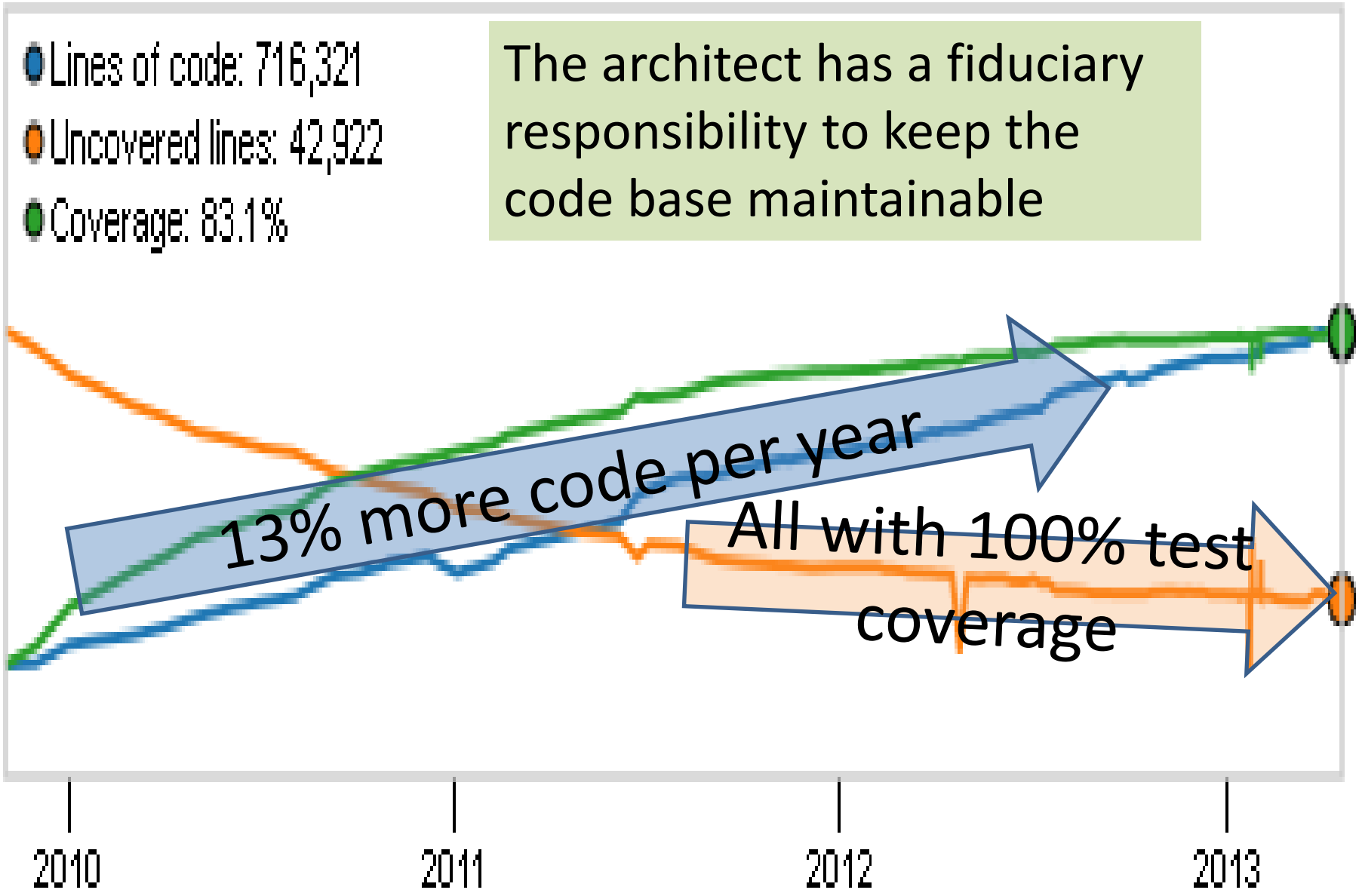
Establish Point of Accountability

Effort breakdown		
Role	task	Total
Code Quality Czar	monitor technical debt	13%
	clear unfixable debt	10%
	mentor/assist	6%
	scoring	5%
	monitor dev branches	4%
	manual db update/fix/check	3%
	ad hoc reporting	2%
	email warning help desk	2%
	standards setting	2%
	investigate numbers/database	1%
Lead developers	report review	16%
	performance appraisal	13%
	presentations	4%
	train	1%
	trend analysis	1%
Code	coding	5%
	write SQL	4%
Manage	awareness	2%
	inform/escalate	2%
	sell	2%
Build	sonar	2%

- 12% of Senior Architect



Maintain Velocity with Zero Debt



What Can You Do Tomorrow?

- Implement a code quality dashboard
  - Sonar
  - Maven code coverage dashboard
  - MS Visual Studio or Eclipse CQ plugin
- Require unit testing
- Peer review at check-in
- Break the CI build if quality decreases
  - Coverage drops below threshold
  - Critical or major static analysis violations