Using TSP to Develop and Maintain Mission Critical IT Systems

Alex Obradovic

9/17/2013

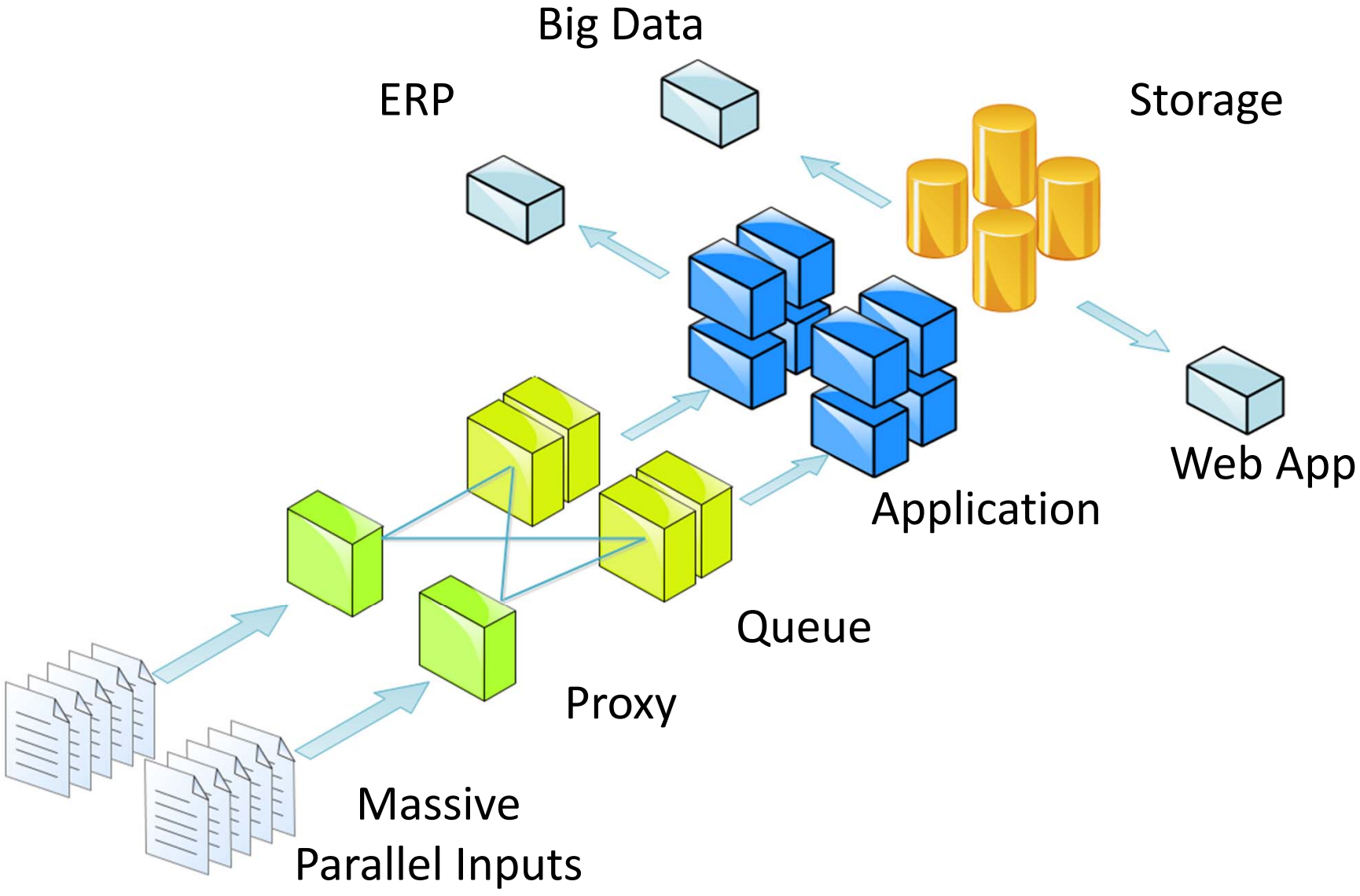
Disclaimer

- The views and opinions expressed in this presentation are those of the author and do not necessarily reflect the official policy or position of Beckman Coulter.
- Examples and analysis within this presentation is based on transient data from 2010-2011 in order to illustrate Software Engineering concepts only, and in no way represent Beckman Coulter product or a service offering.

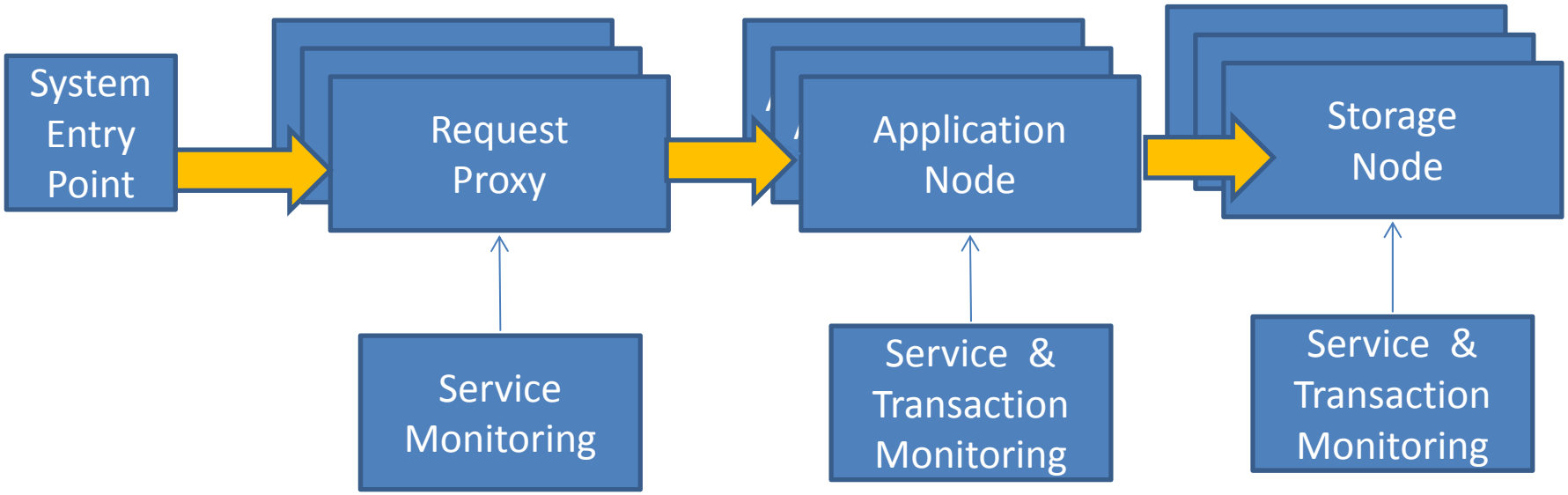
IT/TSP experience

- IT Experience
  - Led teams that develop and support custom built applications
  - Led Global Systems Operations for a large data center in the US
- TSP Experience
  - 2010-2012 TSP Team Lead
  - 2013 TSP Provisional Coach

Data Center Application Diagram



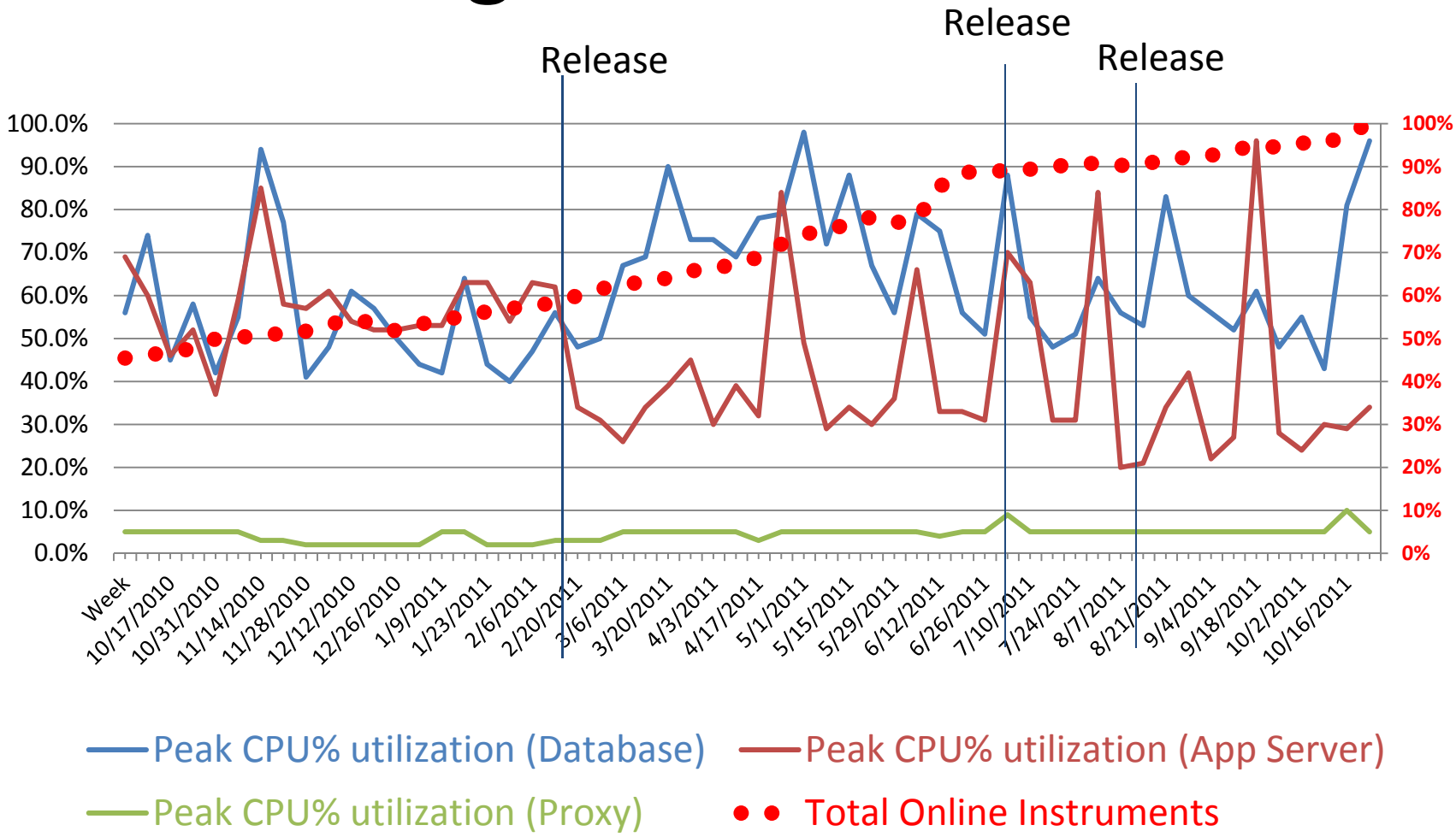
Data Center High Availability/Performance topology



Major architectural drivers for the data center application:

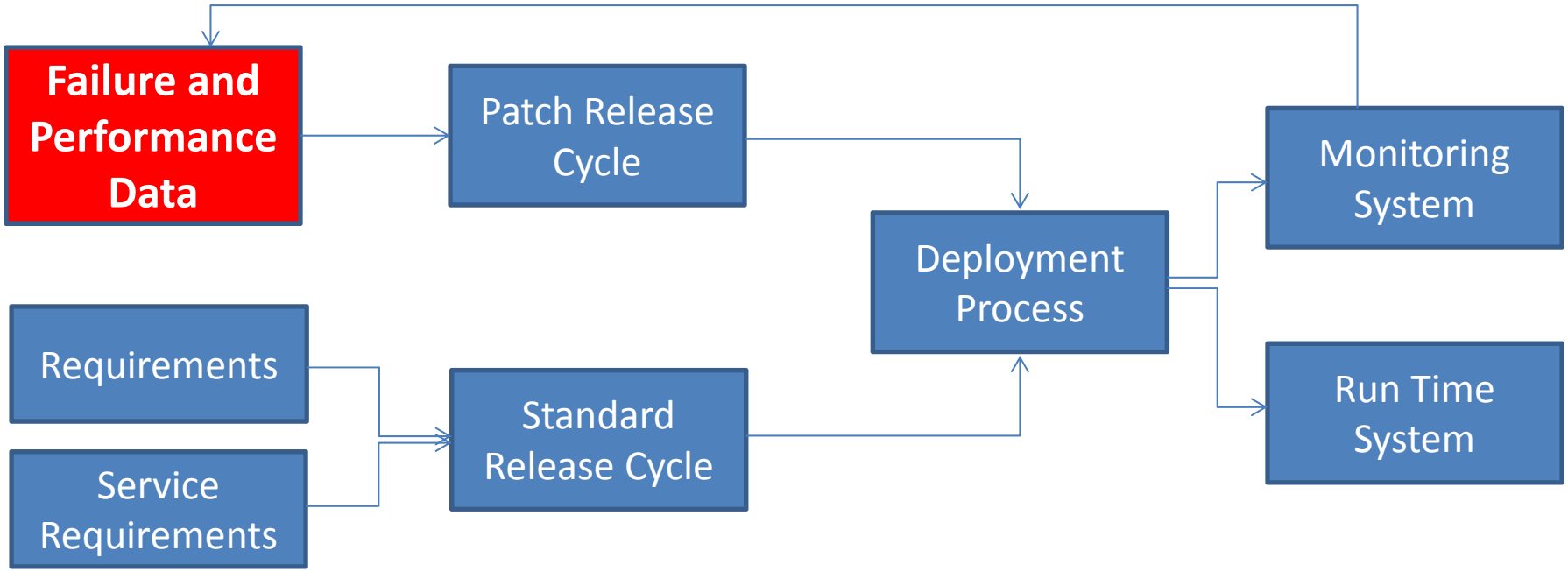
- Availability
- Performance
- Scalability

Monitoring Cluster Performance



Application performance profile changes as the application load increases.

Maintenance/Development Model



The runtime monitoring system provides valuable data that feeds into the software development cycle, and resulting software changes are released either through patches or standard feature releases.

Non-TSP Cycle 0 (Apr 2010-Feb 2011)

	2010									2011	
	Apr	May	Jun	July	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Plan	**Dev**	**Dev**	**Dev**	**Dev**	**Dev**	**Test**					
Actual	**Dev**	**Dev**	**Dev**	**Dev**	**Dev**	**Dev**	**Dev/ Test**	**Dev/ Test**	**Dev/ Test**	**Test**	**Test**

Plan:

- 6 modules
- 6-month release cycle

Actual:

- 6 modules
- 1 new app, 1 maint. release
- 11-month release cycle

The first non-tsp project that I led was late by 5 months. The team missed the delivery date due to scope increase, performance maintenance mini releases, and longer than expected testing cycle.

Non-TSP Cycle 0 Code Metrics

Base:	311,878
Deleted:	590
Modified:	490
Added:	38,622
Added & Modified:	39,112
Total:	349,910
Defects found in systems test:	386
Total non-TSP Defects per KLOC	9.9

The number of defects discovered in systems test prompted us to seek opportunities to reduce rework and lower the cost of development.

Time allocation to fix a single defect

Activity	Hours
Prep for Testing	1
Team Review/Prioritization	1
Testing	1
Logging Operational Defect	0.5
Developer Review	1
Design and Implementation	3
Redeployment of Binaries	1
Unit Testing, Integration	2
Verification Testing	1
Total time to fix a defect in hours	**11.5**

Due to the complexity of the environment it took on average 11.5 hours to detect, log, and fix a defect.

Calculating Cycle 0 Cost to fix all defects

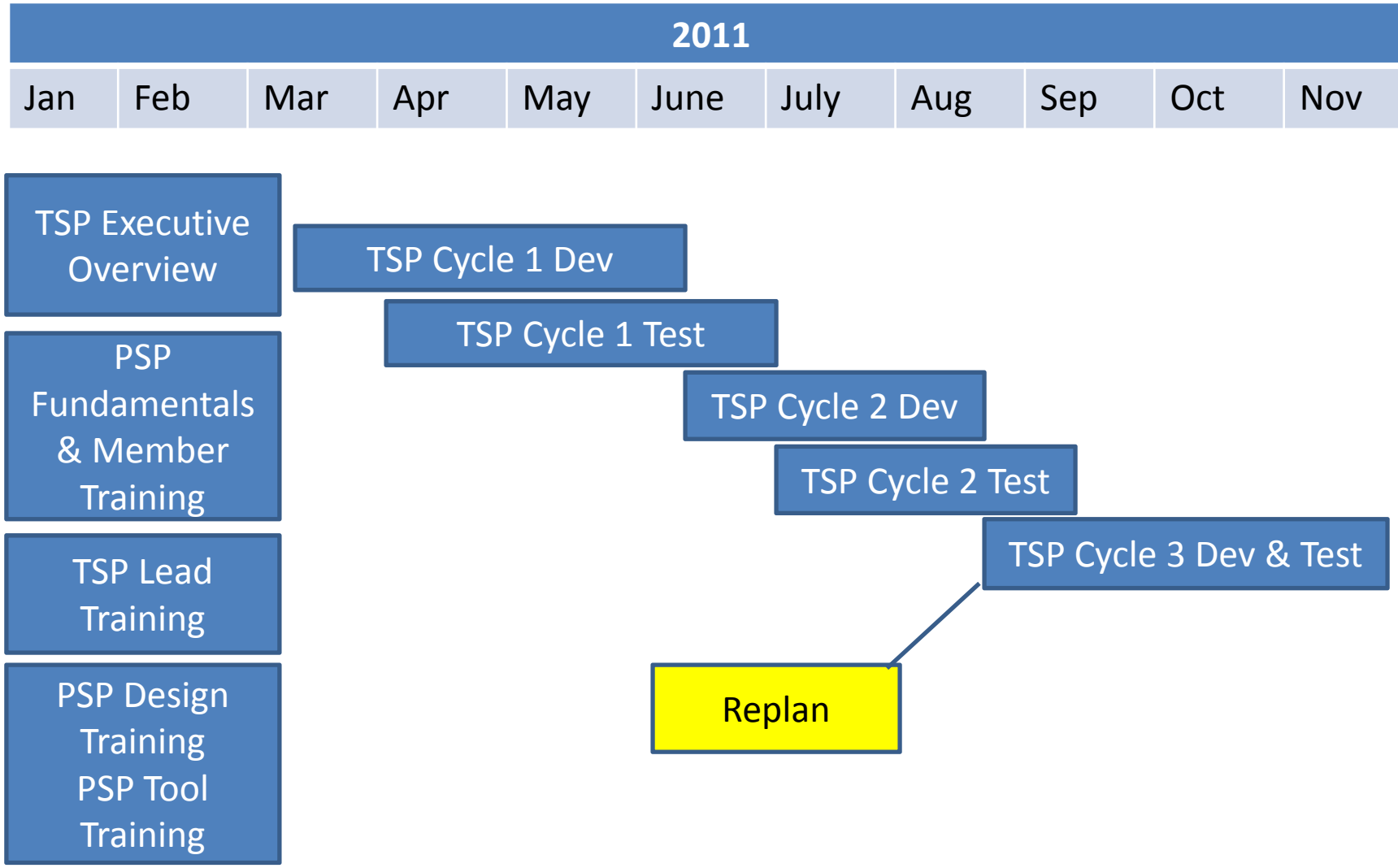
Metric	Non-TSP Cycle 0
New and Modified Lines of Code (LOC)	39112
Actual Defects/KLOC - system test	9.9
Hours to find, fix, and retest a defect	11.5
Estimated Blended hourly rate	$85.00
Total Defects	387
Total hours to fix issues	4451
# of Testers and Developers	16
Non-Admin, Direct project Hours Per week	12.5
Weeks Needed to fix issues	22
Direct costs of fixing defects	~ $378K
Cost of the team per week	~ $54K
Total including admin costs	~ $1.2M

Economics of Quality for Cycle 0 scenario

	Baseline	2x Quality Improvement	6x Quality Improvement
Defects Per KLOC	9.9	5.0	1.7
Defects in Systems Test	386	194	65
Direct Cost	~ $377K	~ $190K	~ $64K
Weeks to fix (16 people)	22	11	4
Admin Cost	~ $1,2M	~ $605K	~ $201K

The team discussed quality improvement opportunities and considered TSP for application development.

Timeline of 3 TSP Cycles in 2011



The team was trained in January/February of 2011 and proceeded with 3 TSP Cycles

2011 TSP Metrics

	Cycle 0 (Non TSP)	Cycle 1	Cycles 2&3
Effort in task hours	Not known	683	904
Lines of Code	39,112	5,091	8,900
Defects per KLOC	9.9	6.9	1.8
Total Defects	386	35	16
Plan Growth (Scope change)	Not known	47%	7%
Effort Overestimation	Not known	13%	11%
Schedule Net Effect	83% over	34% over	-4% (delivered earlier)

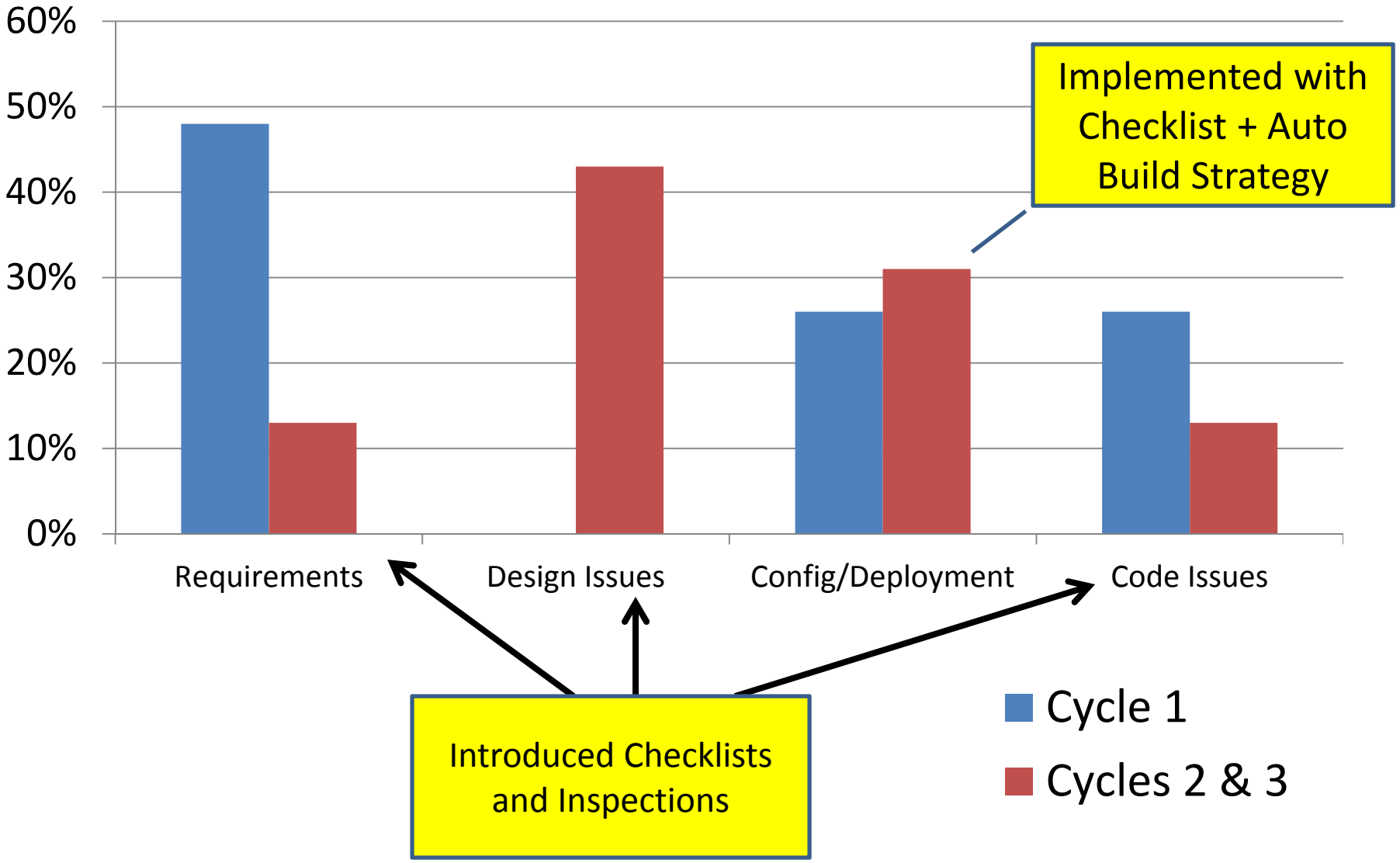
After introducing TSP it was evident that the team has the ability to better control defect rates and schedule scope.

Team Goals

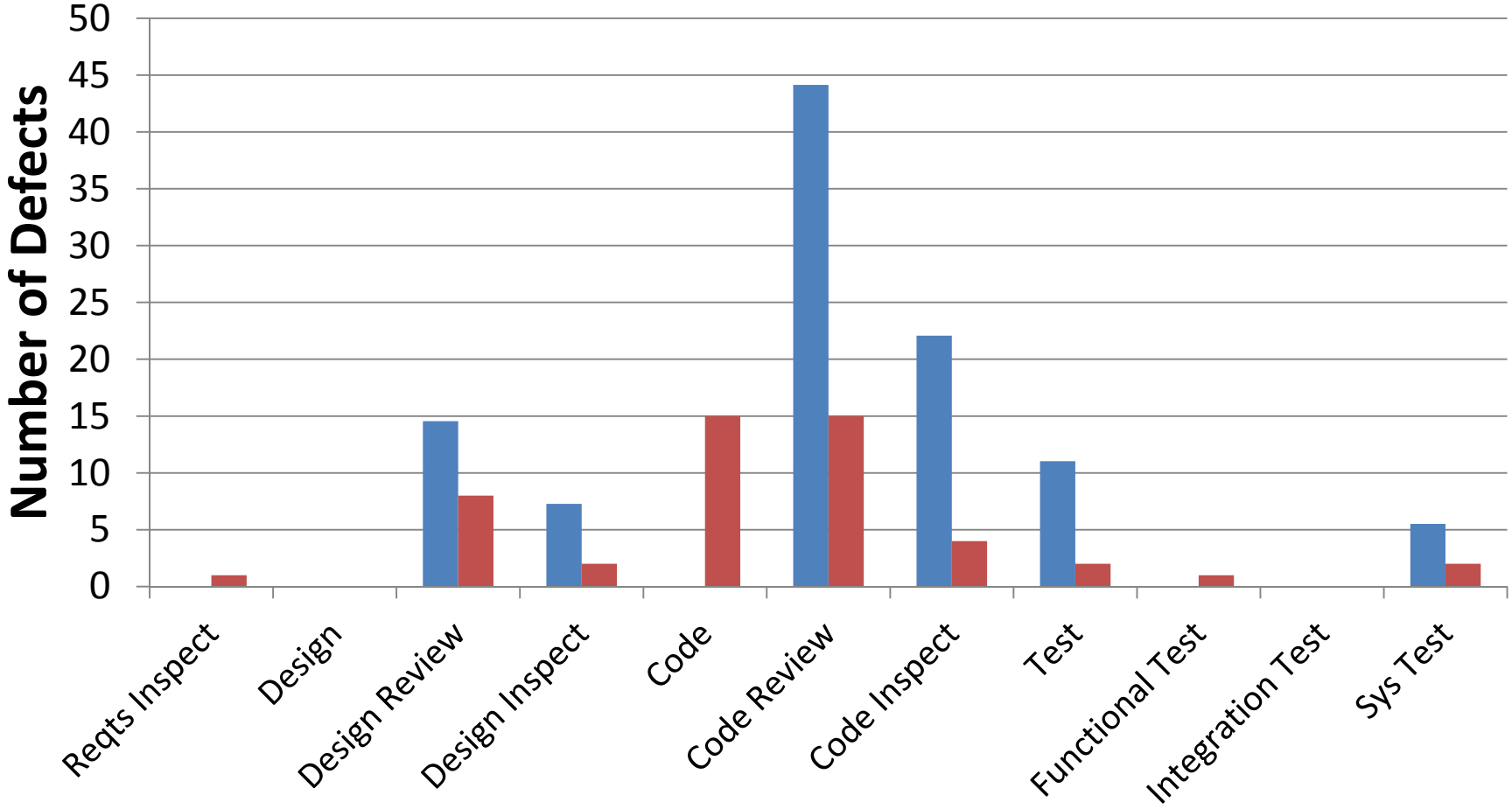
Goal	Cycle 1	Cycles 2&3
System test defect density <2 defects/KLOC	6.09	1.8
Automated Unit Tests and Static analysis (80% code coverage, 0 major exceptions)	Not Met	Met
100% Official Inspection Coverage	Met	Met
Zero defects in system testing	Not met	Not met
+/-10% schedule accuracy	+34%	-4%

The TSP framework enabled the team to set and meet their own quality and schedule goals.

Sources of Defects

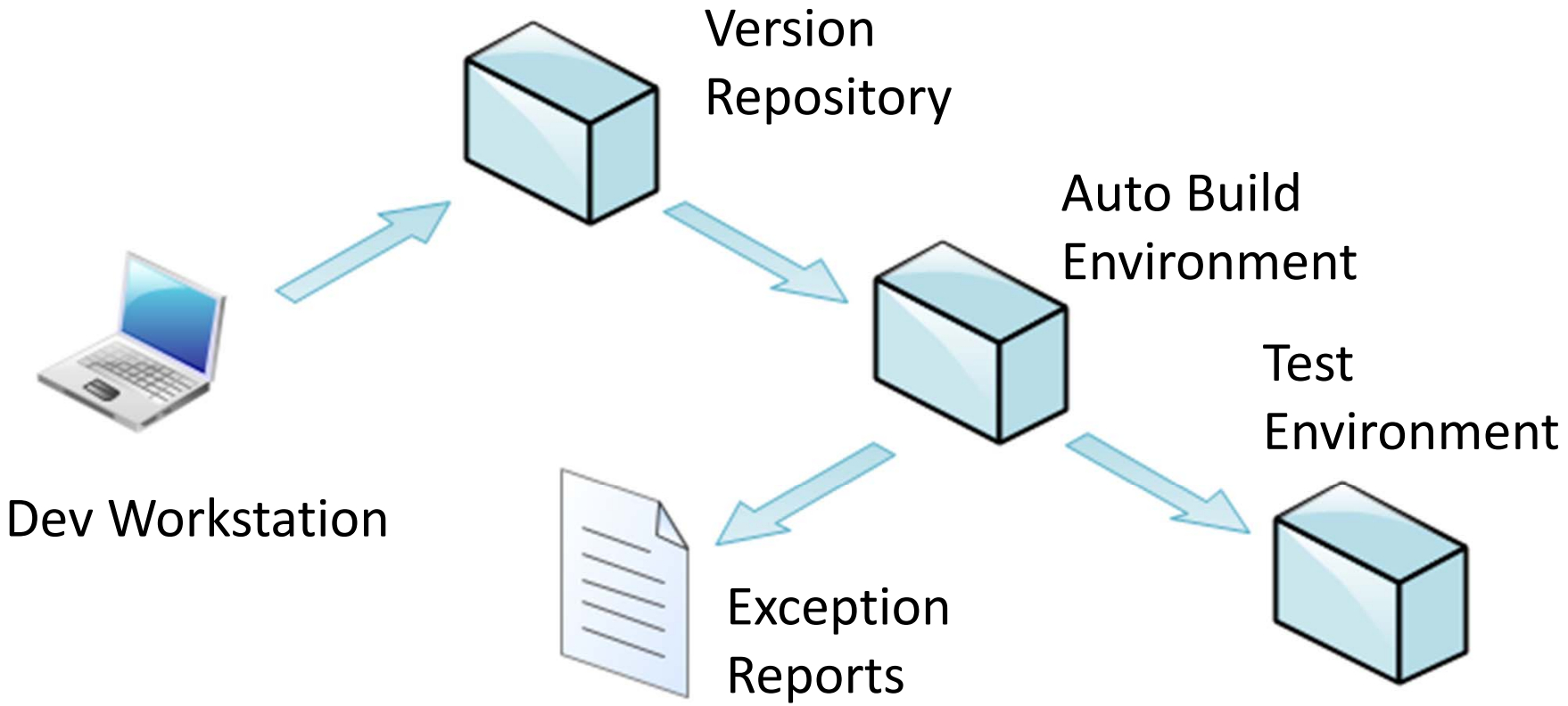


Expected vs. Actual Defects



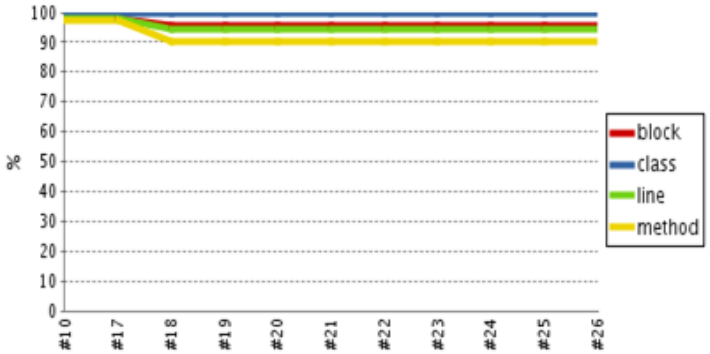
The team became confident that they could remove a large number of defects before system test.

Addressing Configuration/Deployment Issues



Checklists and automation of the build system virtually eliminated configuration and deployment issues.

Automated Unit Test Code Coverage



Coverage Summary

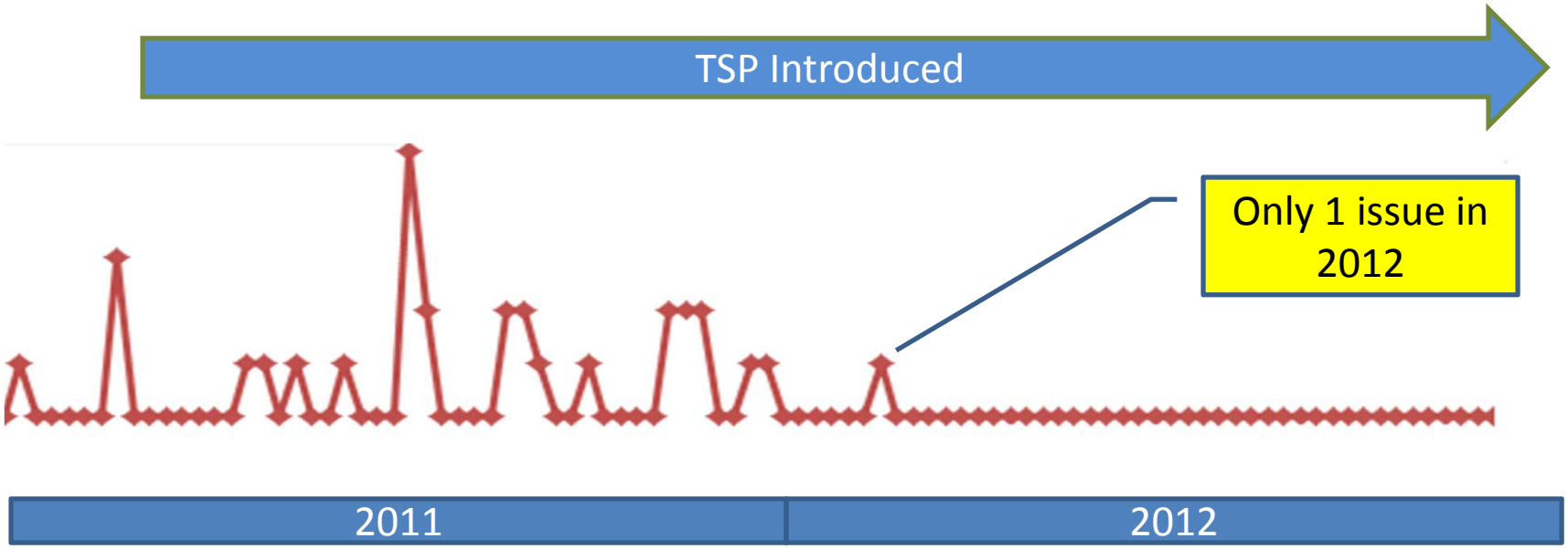
name	class	method	block	line
com.bci.rms.instrument.sms.ea.parser.db	100.0% 6/6	90.2% 37/41	95.3% 1409/1479	94.6% 247/261

Coverage Breakdown by Source File

name	class	method	block	line
ActivitySMSSStartup.java	100.0% 1/1	83.3% 5/6	99.8% 838/840	99.0% 101/102
DBColumn.java	100.0% 1/1	83.3% 5/6	80.0% 36/45	91.4% 13/14
SMSEventLookup.java	100.0% 1/1	100.0% 5/5	89.3% 100/112	93.1% 27/29
SMSEventLookupRecord.java	100.0% 1/1	92.3% 12/13	93.3% 42/45	94.7% 18/19
SMSEventQueryGenerator.java	100.0% 1/1	100.0% 2/2	86.9% 113/130	92.6% 25/27
SMSInstrumentDataQueryGenerator.java	100.0% 1/1	88.9% 8/9	91.2% 280/307	90.0% 63/70

The team used static analysis, automated unit test, and code coverage tools as additional controls to sustain code quality.

Trend of datacenter runtime incidents



Number of data center issues was reduced from 47 in 2011 to 1 in 2012.

* There is no correlation between recorded incidents and service availability due to the high-availability data center design.

Quality cost comparison for Cycles 2&3

	Cycle 0 Quality	Cycle 2&3 Quality
Lines of code	8900	8900
Actual Defects/KLOC - system test	9.9	1.8
Hours to find, fix, and retest a defect	11.5	11.5
Blended hourly rate	$85.00	$85.00
Total Defects	88	16
Total hours to fix issues	1013	184
# of Testers and Developers	8	8
Direct project hours per week	12.5	12.5
Weeks Needed to fix issues	10	2
Direct costs of fixing defects	~ $86K	~ $16K
Cost per week	~ $27K	~ $27K
Total including admin costs	~ $272K	~ $54K

Lessons Learned

- Task hours hovered around 12.5 hours per person per week
- TSP Process can be customized to match the environment: Deployment Process, Production Incident Process, Static Analysis, etc.
- Checklists: Configuration, Deployment
- New design goals (UML tools to provide Internal/External/Dynamic/Static designs)

TSP adoption

Skillsets	Local development team, track 1	Local testing team	Local development team, track 2	Offshore team
Members	4	3	1	3
TSP data collection effectiveness	High	High	Medium	Low
TSP Effective	Yes	Yes	No (Not enough team members)	N/A (Missing a Coach)

TSP was most effective for local teams that had 3 or more team members.

Final Thoughts

- TSP can be very effective in IT environments
- TSP metrics can be used to calculate cost of fixing defects
- Availability of internal organizational support was a key factor for sustaining TSP
- TSP Review and Inspections were key in reducing defects and improving maintainability of code
- Splitting feature development and maintenance teams may be necessary for dynamic IT environments

Challenges

- Team member changes may impact team and TSP effectiveness.
- Maintaining TSP project focus may be difficult if the team members work on multiple non-TSP projects.
- TSP adoption is difficult for single or two-person teams or virtual offshore teams
- Resource allocations/Cost constraints
- May be difficult to establish TSP momentum initially
- Transitioning from a team lead to a TSP Coach

Thanks

- Team
- Coaches, Instructors, Mentor Coach
- IT and R&D
- Beckman Coulter

Questions?

Email: aobradovic@beckman.com