SADIO Electronic Journal of Informatics and Operations Research

http://www.sadio.org.ar

vol. 10, no. 1, pp. 108-122 (2011)

Improving Product Quality and User Satisfaction through Early Customer Feedback and Automation

Matias Cabral[1] Domingo Gonzalez[1] Dan Hirsch[1]

Cesar Martinez[1] Andres More[1] Victor Rosales[1]

[1] Intel Software Argentina - Argentina Software Development Center (ASDC).
Corrientes 161 – 2nd floor
X5000ANC, Córdoba, Argentina
e-mail: {matias.a.cabral, domingo.c.gonzalez, dan.hirsch, cesar.martinez, andres.more, victor.h.rosales}@intel.com

Abstract

New users of cluster computing find a high entry barrier due to the daunting complexity of deploying and managing a cluster. The Intel® Cluster Ready program aims to lower such entry barrier by setting standards for the cluster integration process and providing tools to simplify the deployment process. The Intel Cluster Checker tool, a key component of the Intel Cluster Ready program, is an automated and flexible tool that validates the cluster settings against the Intel Cluster Ready specification, and checks the general wellness of the cluster. This experience report shows how the product engineering team of Cluster Checker faced the challenge of having a small team and be able to provide a cutting-edge high-quality product in less time, maximizing resource usage through automation, and increasing and securing early validation with key stakeholders..

Keywords: High Performance Computing, Automation, Development Process, Agile Methodologies.

1 Introduction

The *Intel® Cluster Ready Program* [1] aims to reduce the time-to-market and minimize the risk involved in selecting a collection of hardware and software components for High Performance Computing environments, providing a thoroughly tested solution stack and ensuring the interoperability of its components out-of-the-box.

Key to this program, the Intel Cluster Checker [1] is a powerful and flexible tool for helping customers to integrate clusters by validating the compliance of a new cluster solution with the Intel® Cluster Ready Specification [1]. The product includes 31K lines of source code distributed in more than 120 independent test modules, executing over +10 officially supported operating systems. Although Independent Product Validation is done using four 4-node cluster environments over the latest Intel and third-party technologies, the actual testing goes beyond that, covering cluster scale-outs up to 256 server nodes; heterogeneous server platform combinations; and pre-release hardware and software component compatibility (both, for Intel and third parties) to support the time to market launch of Intel's new technology.
Furthermore, as the Intel Cluster Checker is integrated by the program partners (Original Equipment Manufacturers, Platform Integrators and channel members) in their cluster manufacturing processes, there is a need for the tool engineering team to efficiently and timely address partner requirements that are critical for their business continuity and the launch of their own Intel Cluster Ready certified products.

Agile Methodologies require a fast response to all issues like a sudden change in requirements, bug fixing, or implementing new features. This allows the team to continuously deliver a working product with the certainty that it is what the user wants and in a short period of time (usually no more than 3 weeks). But regardless of the methodology used, all quality goals must always be reached. This experience report shows how the Intel Cluster Checker engineering team, located at Intel's Argentina Software Development Center, has implemented reliable, efficient and high quality processes and infrastructures, coping with the complexity of the validation space and the critical time to market business requirements, through the integration of Agile development methodologies with a *CMMi* process initiative, plus the end-to-end automation support of the development process.

2 Background

2.1 Intel Cluster Ready Program

In recent years, cluster computing has emerged as a scientific tool to obtain additional computational power in the commercial (manufacturing and services), health, finance, and educational areas [2]. Furthermore, as Figure 1 shows, the cluster market is growing faster than the non-cluster servers market [5]. It is estimated that the High Performance Computing (HPC) server market revenue will grow past $12 billion in 2013, from $9 billion in 2009, with clusters making up for more than 70% of this amount [6].

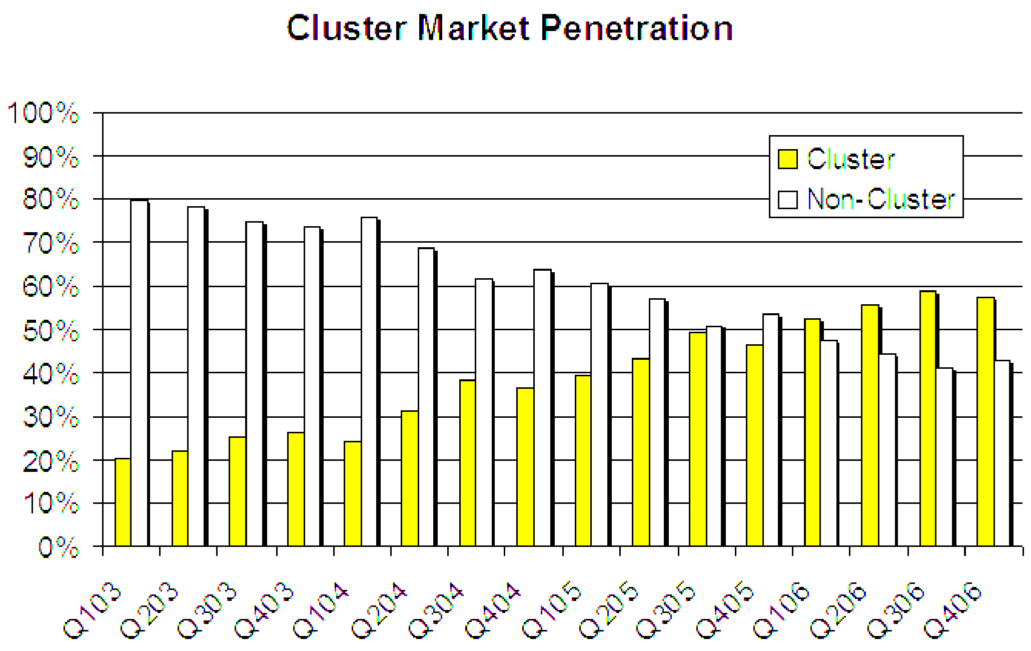


Figure 1. Cluster vs. Non-Cluster Server Market Penetration

The comparatively low cost and scalability of clusters is pushing more companies to enter in the cluster's arena. However, the capability of each cluster involves an exchange of hardware costs for software costs [7]. A complex combination of software is required to configure and maintain multiple distributed machines that make a single clustered system. Therefore, new users are forced to do extensive research to establish hardware and software requirements, build the cluster, install the required software, and then tune the components to obtain a healthy system with a competitive performance. Given the complexity of building a cluster, different users often develop ad-hoc procedures for automated cluster deployment and management. The associated lack of standards have kept the cluster ownership costs high as custom procedures should be developed from scratch on virtually every installation. To ameliorate this situation, some of the "best practices" techniques for cluster deployment were integrated in easy-to-use toolkits called provisioning systems [7,8]. However, the solution offered by provisioning systems only considers certain software stacks, hopefully over a small set of hardware platforms.

In order to overcome the comp lexity of this environment, Intel developed the *Intel Cluster Ready Program* [1] to simplify the design, building, and deployment of clusters.

With more than 100 partners around the world, the Intel® Cluster Ready Program is a collaborative effort between Intel, Original Equipment Manufacturers (OEMs), Platform Integrators (PIs), channel members, and Independent Software Vendors (ISVs), that signup as Intel Cluster Ready partners, in order to establish an architecture and specification to be used as a commo n basis for performing clusters. Cluster solutions that comply with the Intel Cluster Ready Specification [1] are certified by Certification Authorities of the Intel Cluster Ready program. Also applications produced by ISVs can be registered as compliant with the Intel Cluster Ready specification.

The main idea behind the Intel Cluster Ready certification process it that applications written to run on one certified cluster can run on any certified cluster with a high confidence. Conversely, a certified cluster will be able to run any registered application without requiring major efforts.

To complete the certification of a cluster solution, the Intel Cluster Ready partner (OEMs, PIs, channels) may follow the steps indicated by a Cluster Reference Implementation (a.k.a. recipe) [1]. These recipes are the product of a cluster engineering process that will allow the Intel Cluster Ready partner to manufacture compliant clusters. The next step in the process is to use the Intel Cluster Checker Tool to automatically verify the cluster health and compliance with the Intel Cluster Ready Specification. The tool used for the automatic verification is the Intel Cluster Checker, which is introduced in the next section.

2.1 Intel Cluster Checker Tool

The *Intel Cluster Checker* [1] is a software tool that helps the verification of cluster compliance with respect to the Intel Cluster Ready Specification. It can verify the cluster at the node and cluster-wide levels, ensuring that cluster nodes are uniformly and optimally configured. The tool is customizable and extensible, letting users develop their own tests by specifying commands to be executed and their expected results.
It is a Perl program with over 30K lines of code that is distributed in binary form. It has over 120 different tests, with an average of 6 configuration items for each one of them. It has almost 300 pages of documentation that are distributed in PDF, HTML and Man-Page format (that is a lmost 1000 pages).

In a nutshell, it consists of an engine that loops over a set of test modules (see Figure 2). Prior to entering the loop, the engine must figure out which test modules to run, which nodes to check, and setup the infrastructure to process the test modules.

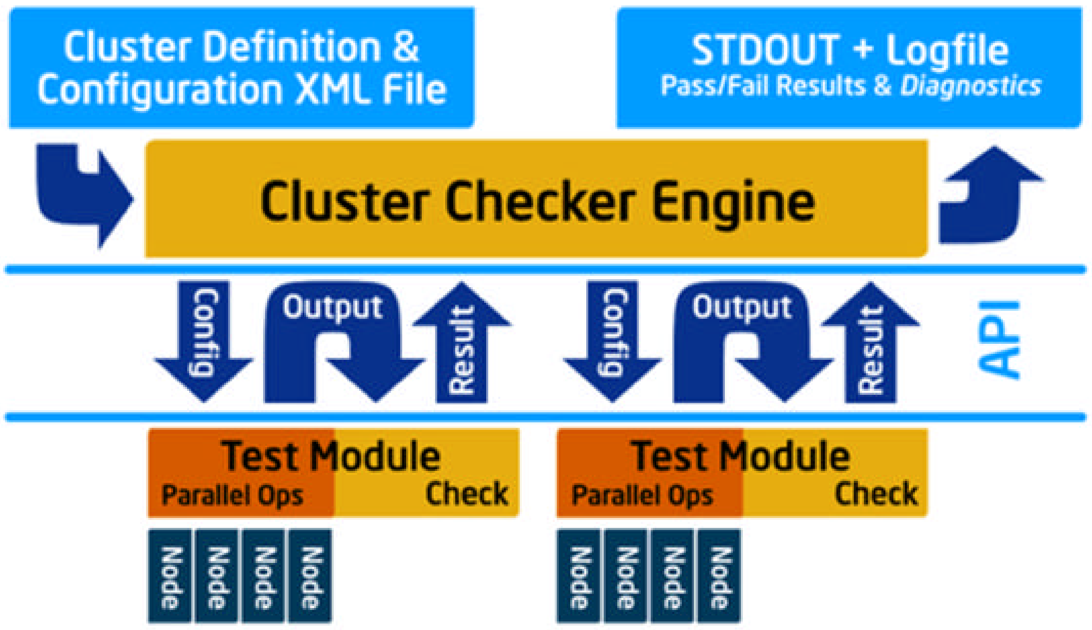


Figure 2a. Test Modules Execution flow.

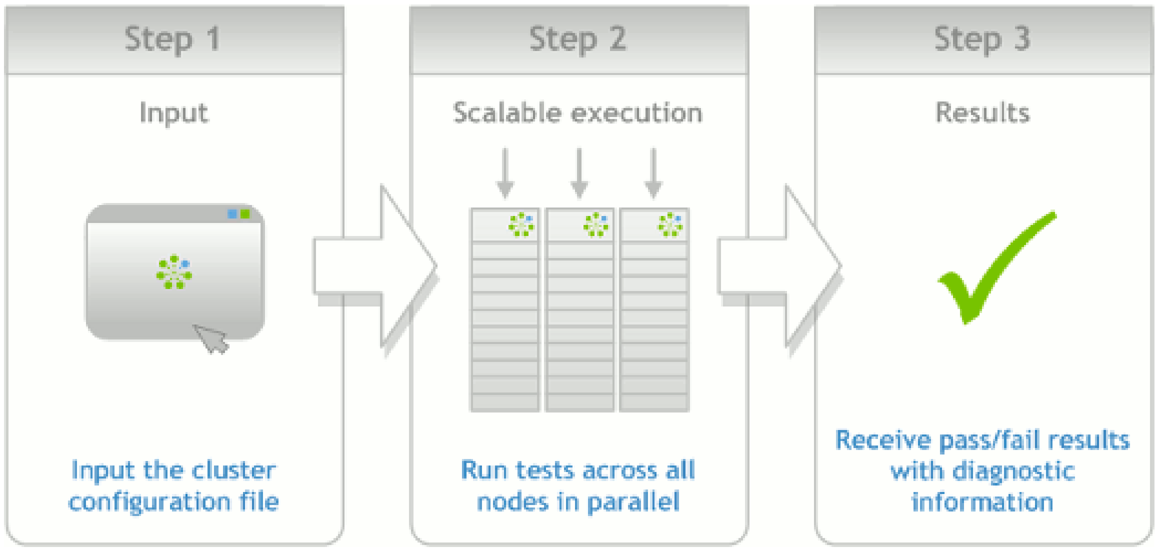
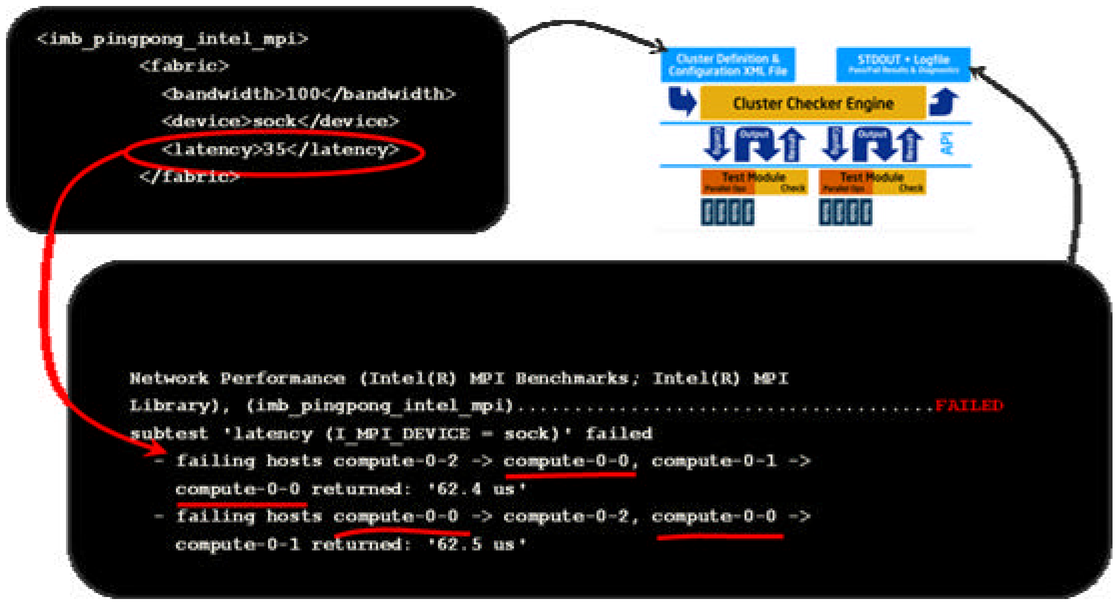

Step 1
Input
Input the cluster configuration file
Step 2
Scalable execution
Run tests across all nodes in parallel
Step 3
Results
Receive pass/fail results with diagnostic information

Figure 2b. Cluster Checker process workflow.



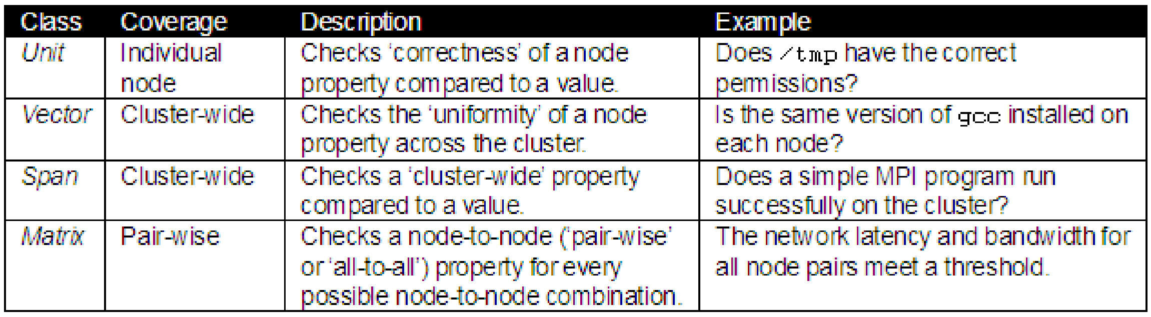
```
<imb_pingpong_intel_mpi>
        <fabric>
          <bandwidth>100</bandwidth>
          <device>sock</device>
          <latency>35</latency>
        </fabric>
```

```
Network Performance (Intel(R) MPI Benchmarks; Intel(R) MPI
Library), (imb_pingpong_intel_mpi)...................................FAILED
subtest 'latency (I_MPI_DEVICE = sock)' failed
   - failing hosts compute-0-2 -> compute-0-0, compute-0-1 ->
     compute-0-0 returned: '62.4 us'
   - failing hosts compute-0-0 -> compute-0-2, compute-0-0 ->
     compute-0-1 returned: '62.5 us'
```

Figure 2c. Test module configuration and result examples.

The modules themselves do not execute on the cluster nodes; a module runs one or more commands on a node. So, there is no needed for any specific support at each node of the cluster. A module has two functional steps: it first collects information and then determines if the information is correct. Correct may mean matching a certain value, uniformity compared to other values, or something else. Thus, modules must have a gather step where it collects information and a test step where it determines if that information is correct.

As shown in Figure 3, there are four different classes of modules to handle special cases of this two-step gather/test p rocess, but all classes respect the aforementioned structure.



Class	Coverage	Description	Example
Unit	Individual node	Checks 'correctness' of a node property compared to a value.	Does `/tmp` have the correct permissions?
Vector	Cluster-wide	Checks the 'uniformity' of a node property across the cluster.	Is the same version of `gcc` installed on each node?
Span	Cluster-wide	Checks a 'cluster-wide' property compared to a value.	Does a simple MPI program run successfully on the cluster?
Matrix	Pair-wise	Checks a node-to-node ('pair-wise' or 'all-to-all') property for every possible node-to-node combination.	The network latency and bandwidth for all node pairs meet a threshold.

Figure 3. Classes of Modules.

There are two types of checking: the wellness checking of a cluster and the compliance mode with respect to the Intel Cluster Ready Specification.

Wellness checking verifies the functional and non-functional characteristics of a cluster, like performance or disk space. It can be used for on demand validation of the cluster. This feature of Intel Cluster Checker helps the maintenance of the cluster health providing diagnostic data to identify the issue.

Compliance checking is used during the Cluster Certification process of the Intel Cluster Ready Program. The goal of applying this checking is to determine if the cluster configuration follows the Intel Cluster Ready Specification.

3 Agile Methodologies and Automation support

As we have mentioned in the introduction, although Independent Product Validation of the Intel Cluster Checker is done using the four top 4-node cluster environments, the actual testing goes beyond that, covering cluster scale-outs up to 256 server nodes; heterogeneous server platform combinations; and pre-released hardware and software component compatibility (both, Intel and third party) to support the time to market launch of Intel's new technology. Also, as the Intel Cluster Checker is integrated by the program partners in their cluster manufacturing processes, there is a need for the development engineering team to efficiently and timely address partner requirements that are critical for their business continuity and the launch of their own Intel Cluster Ready certified products.

The product is made up of 31K lines of source code distributed in more than 120 independent test modules, executing over +10 officially supported operating systems. Currently, the product is being developed at the *Argentina Software Development Center (ASDC)* in Córdoba. The engineering team consists of 2 developers, 1 tester and 1 independent product validation engineer.

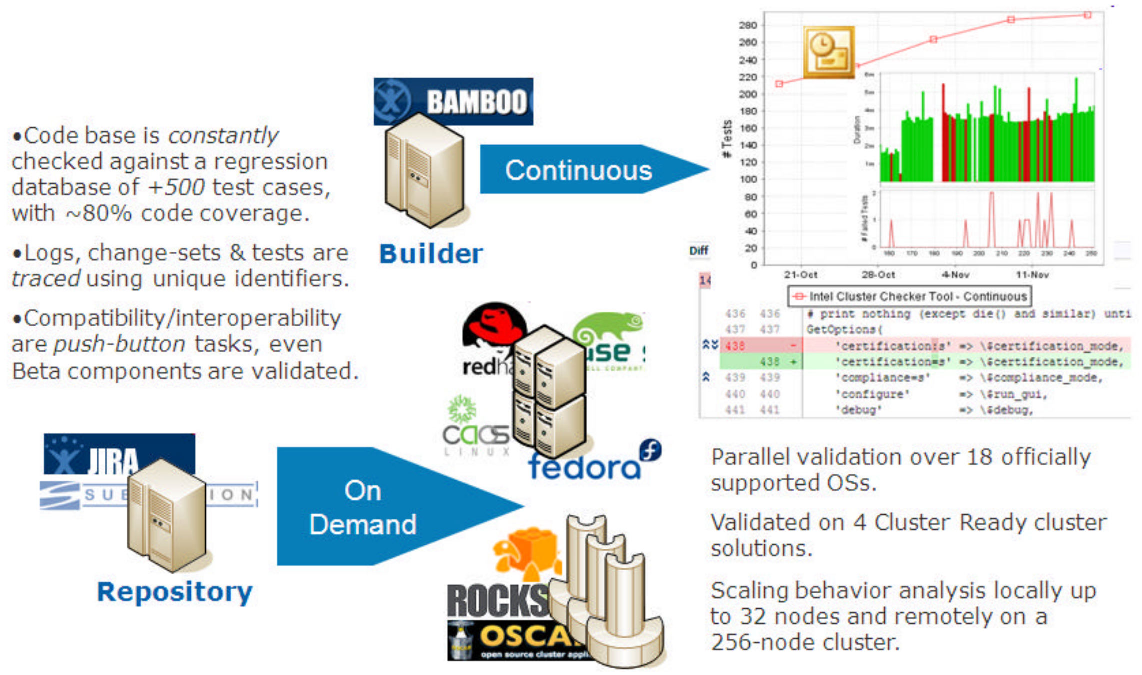
The main challenge faced by the reduced engineering team has not been only to provide a cutting-edge high-quality product in less time, while maximizing resource usage, but also to increase and secure early validation with key stakeholders. Moreover, last minute requirements requested by Intel Cluster Ready partners should be included based on their feedback.

Cluster Checker has an extremely wide range of requirements. The tool has to successfully execute on multiple hardware platforms, software stacks and over many OSs (Operative Systems). Some of these requirements grow in time by their nature, such as supporting other Intel tools (compilers, math kernel libraries, etc), supporting new OSs versions and distributions, and others. And all these should include backward and forward compatibility, including support of older versions of the ICR specification. Several non-functional requirements also need to be met, like execution time reduction critical to allow cluster integrators and manufactures to include Cluster Checker as part of their manufacturing processes. A second related requirement is scalability which manages execution time explosion while allowing expansion over clusters of bigger size (i.e. thousand of nodes). So, even without adding any new feature an enormous amount of effort is required to fulfill these persistent requirements. Summing up, without concise processes, smart automation and well established policies, fulfilling the requirements, maintaining and improving the tool simply cannot be achieved by a small team.

To achieve these goals, the engineering team has been driving process and infrastructure improvements over the last year and a half, involving three releases of the tool. One of the main focuses of these improvements has been to provide automation support throughout the different tasks they performed.

3.1 Infrastructure

On the infrastructure side, the project has included an aggressive effort for building an automated development and testing infrastructure (Figure 4), avoiding recurrent effort on repetitive manual tests, while supporting the required software and hardware coverage against a small time to market window.


•Code base is *constantly* checked against a regression database of +*500* test cases, with ~80% code coverage.
•Logs, change-sets & tests are *traced* using unique identifiers.
•Compatibility/interoperability are *push-button* tasks, even Beta components are validated.
BAMBOO
Builder
Continuous
JIRA
Repository
On Demand
Diff
Intel Cluster Checker Tool - Continuous
# Tests
21-Oct
28-Oct
4-Nov
11-Nov
Parallel validation over 18 officially supported OSs.
Validated on 4 Cluster Ready cluster solutions.
Scaling behavior analysis locally up to 32 nodes and remotely on a 256-node cluster.

```
# print nothing (except die() and similar) unti
GetOptions(
-    'certification:s' => \$certification_mode,
+    'certification=s' => \$certification_mode,
     'compliance=s'    => \$compliance_mode,
     'configure'       => \$run_gui,
     'debug'           => \$debug,
```

Figure 4. Development, Testing and Validation Infrastructure.

The engineering infrastructure includes 2 servers (based on different platforms) for development, 4 clusters dedicated to testing (15 nodes on 4 different server platforms), and 1 server with 18 virtualized GNU/Linux distributions for automatic operating system testing coverage. For product validation the team uses, on demand, a laboratory of more than 300 servers that allows up to 32-nodes cluster scale-outs over 4 different multi-core and multi-processor Intel platforms; and remote access for scale-out validation up to 256 nodes. This infrastructure allows covering a wide range of the key hardware and software stacks for High Performance Computing.

The development and testing teams rely on several tightly coupled tools and a flawless working environment for its day-to-day work. All issues are tracked in a dedicated tool with the information needed for their planning and solution. All configuration items are properly stored under a SCM (Software Configuration Management Plan) to ensure a rollback capability so all reported issues can be traced back to the source, and to have a tagging mechanism in place. To further ensure complete traceability and isolation, each issue implemented is developed in a dedicated branch with a unique identifier that matches the original requirement and keeps the trunk as stable as possible. All issues have a source code branch and a unit tests branch, so the unit tests can also be tracked.

Once the development of an issue has finished, a script is used to ensure that all code is up to date triggering a regression framework with over 500 test cases and a static code analyzer. The script makes sure that the branch is up-to-date and it generates a detailed report that is sent to the testing team. The testing team will not accept the fix unless the quality of the new code is greater or equal than the previous one, at least one unit test has been added to the framework, a code peer review was performed, and the regression framework has passed 100% of the tests. This testing cycle is performed over the issue branch and once it is finished, a testing report is sent back to the developer, who then merges the branch back into the trunk. To cover the hardware requirements, the quality team has 8 dedicated clusters with different software stacks recreating end users' environments. To increase productivity a script is used by the quality team to execute all test cases in parallel over all clusters.

When the developers merge back their changes to the trunk, an automated building tool triggers the unit test framework against the trunk making sure that the merging did not break anything and assuring a stable code. If a unit test fails an email report is sent to everyone in the team, and it is the developer's responsibility to fix this issue. A nightly build plan gets executed which runs the regression framework over the trunk and builds the product leaving it ready for distribution. Then, the regression framework is ran over the binary package generated in the previous step and, if the build succeeds, a new build plan is executed exercising the binary package in parallel over 18 virtual machines with all supported operating systems, ensuring binary compatibility.

This plan triggers a script in a dedicated environment specially developed to ensure the binary compatibility of the tool across all supported OSs. The *Binary Compatibility Environment* consists on a set of VMs (Virtual Machines), each with a different supported OS (currently 18 OSs according with the approved support plan), and a dedicated VM acting as a proxy node for all other VMs. When the binary plan starts, the script turns on all VMs. As VMs have non-persistent disks, they always appear as a fresh installation of the OSs. From the proxy VM, the binary package, created in the nightly build, is copied to all other VMs and Cluster Checker is executed with a minimal workload analyzing the execution results for any failure. If a failure is found a report with all the details is sent to the team. To ensure a repeatable environment, all the VMs are safely stored under EC (Engineering Computing) infrastructure. At this point the nightly package is ready to go to Product Validation.

3.2 Processes

Based on the infrastructure improvement and automation, the key component that allowed the team to achieve their goals has been the changes applied to the Product Development Process through the implementation of the Capability Maturity Model® Integration (CMMi) [3] - Level 2 and Level 3 Best Practices (as part of a certification strategy at the ASDC center) combined with the adoption of an agile development methodology [4], concretely the iterative incremental development process called SCRUM [9] (see Figure 5). This revolutionary change, combining the CMMi process improvement model and an Agile development methodology, was fully implemented for the last three releases (versions 1.3, 1.4 and 1.5) of the Cluster Checker Tool, delivering a working and validated product after each development sprint.

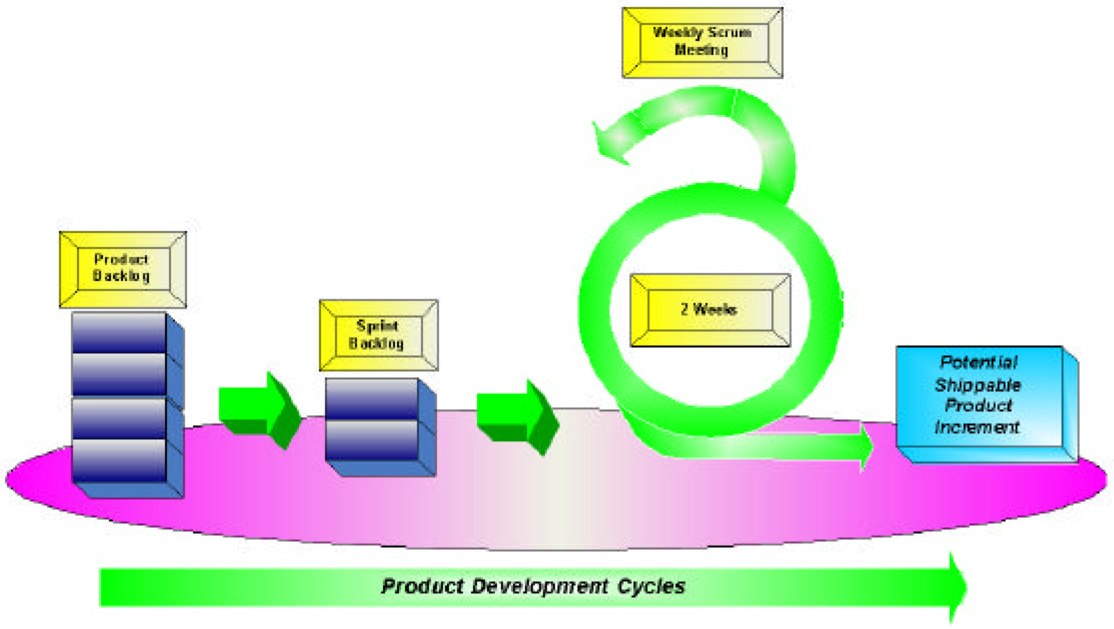

Weekly Scrum Meeting
Product Backlog
Sprint Backlog
2 Weeks
Potential Shippable Product Increment
Product Development Cycles

Figure 5. Development Methodology.

The different engineering areas of the team, that is, Software Development, Testing, Product Validation and Product Packaging, worked together to improve product quality through the implementation of an innovative process to increase automatic testing coverage while keeping track of all documents/source code revisions. All of these enhancements were successfully validated by internal and external audits performed by the *Process and Quality Services* area of ASDC and aligned with the CMMi process initiative in place.

The Product Development Process to build the product was completely migrated from a traditional Waterfall to a *SCRUM* agile software development process. Moreover, it is worth noticing that the release approval process followed the standard Intel approval product lifecycle for a release. As a first step to integrate these processes, the Project Manager in charge was certified as *SCRUM MASTER* to be able to oversee all the procedure. The process started when the product version was approved at *Implementation Plan Approval* phase by the *Software Product Planning Committee* of the *Developer Product Division* of Intel (including the complete Product Requirements Backlog prioritized).

The agile process framework, defined by the software engineers, established a set of *Development Sprints* where the *Change Control Board* committee defined which features are going to be delivered at the end of each sprint. Once a sprint achieved all the pre-established quality targets, the product was distributed to key stakeholders with the purpose of getting early feedback on the features committed for that phase. Each Sprint cycle is executed every two and a half weeks in order to keep the key stakeholders warm and involved in the progress of the tool. Additionally, a closure meeting is held after each sprint by the *Change Control Board* committee to check status, implement any requirement changes and define next sprint planning. Due to the very aggressive schedule and changing requirements, the process requires a high automation level through all the development, testing and product validation phases. Therefore, the set of new test cases and regression test cases have been continuously integrated into an automatic Quality Assurance framework which is executed against each new change, validating the pass/fail rate criteria established at the beginning of the development phase. This setting allows the engineering team to have a constant and hard tracking on the product development behavior; maintaining an always working product state.

As scalability is a critical requirement for the tool, every sprint closure a remote cluster is used to execute Cluster Checker over an increasing number of nodes up to 128 where all execution times and results are gathered and added into a database for further analysis.

Securing a tight alignment with the rest of Intel tools, every release, a recurrent issue is scheduled for updating all distributed binaries compiling them with the latest versions of the tools. Furthermore, all these binaries are tested in the *Binary Compatibility Environment* mentioned in the previous section. Another recurrent issue is to add new VMs to the Binary Compatibility Environment if the OSs support matrix changes, or if a new version of the already supported ones is released (this can be new service packs, or simply minor updates).

A light Product Validation is performed every sprint closure that ensure a proper integration of all issues. Also each sprint release is sent to the Technical Consultant Engineers that are the point of contact with the end users to test the releases or drive pilots with selected end users. In this way early feedback is obtained and it is introduced in the next sprint planning.

Finally the RTM (Ready to Manufacture) package of the last sprint goes to a full Product Validation cycle that guarantees users requirements were fulfill.

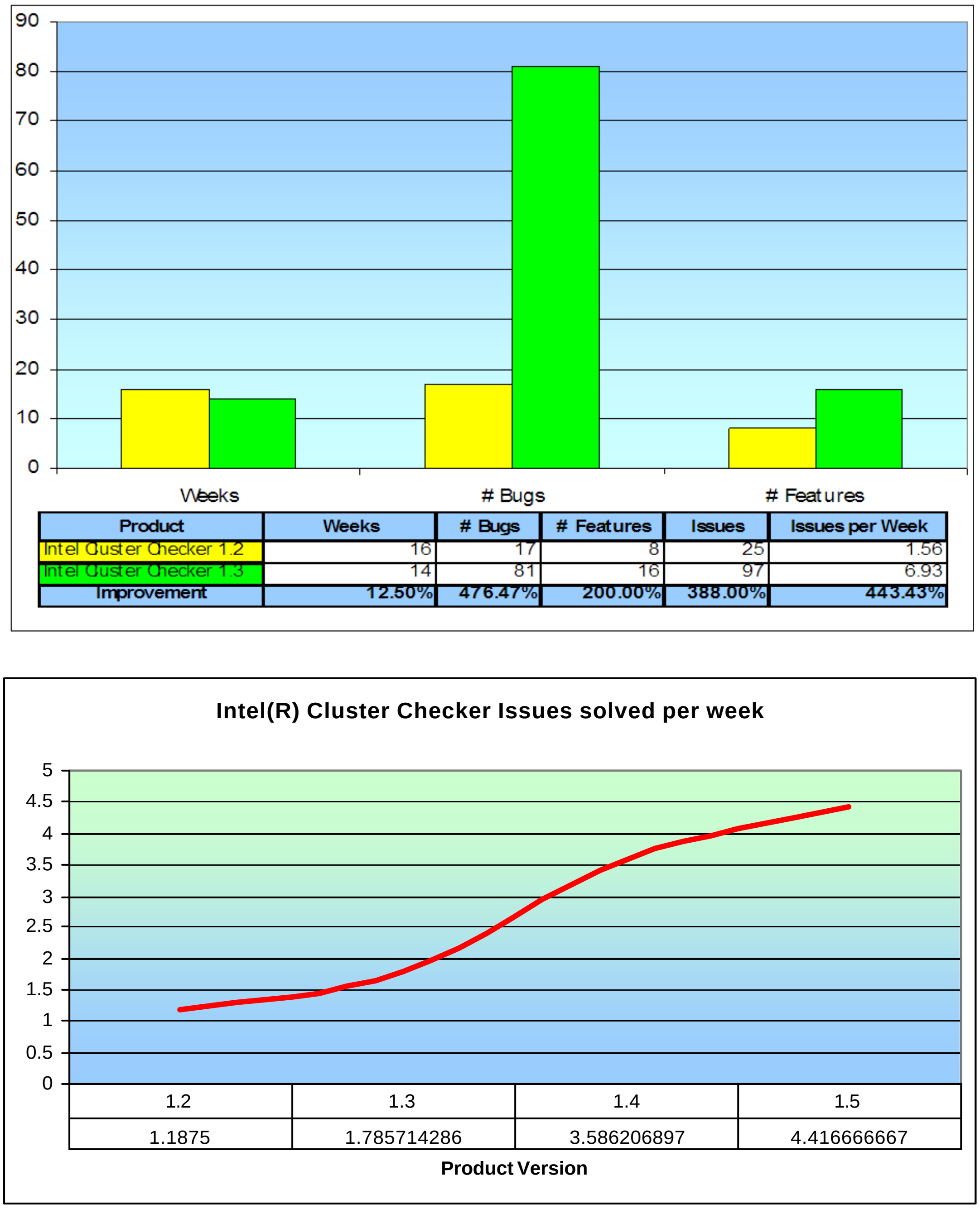
4 Results and Conclusions

Early customer feedback and testing automation allowed Intel Cluster Checker last three versions (versions 1.3, 1.4 and 1.5) to be released achieving sustained higher quality and in a time frame shorter than previous versions. As a result of the process started in Cluster Checker 1.3, the product had key stakeholders validating not only all the requested features but extra features additional to the originally planned ones (Figure 6).
In cold numbers, the process transition showed an initial 12.5% effort reduction, a 400% improvement in backlog resolution, and 200% on new features included in the release cycle (Figure 6).

Following a quality-oriented approach, the size of the automated test case framework was increased from 80 to 500 test cases which were enforced after each code change, resulting in a constant 80% global coverage in the current release version 1.5 (Figure 7), presented through a dynamic drill-down web report which identifies gaps and directs next steps (Figure 8). As part of this, to ensure proper code coverage, the team enforced a policy that the developers must ensure a minimum of 80% of code coverage by creating unit tests for each issue implemented. Furthermore, static source code analysis over both source code and documentation were integrated to guarantee that all known coding pitfalls are avoided and also documentation formatting is homogeneous. The great level of automation involved in every step of the development cycle allows the team to focus on the implementation and further improvements of the tool increasing its productivity.

Automation focus and moving from a typical waterfall development process to an agile one, improved the accuracy of the implementation with respect of the requirements. A 2.8x increase of code coverage and a 6.2x increase in the number of unit tests, the whole automated environment and all the processes in place made a 3x improvement in issues solved by week, 4.2x improvements in codes quality.

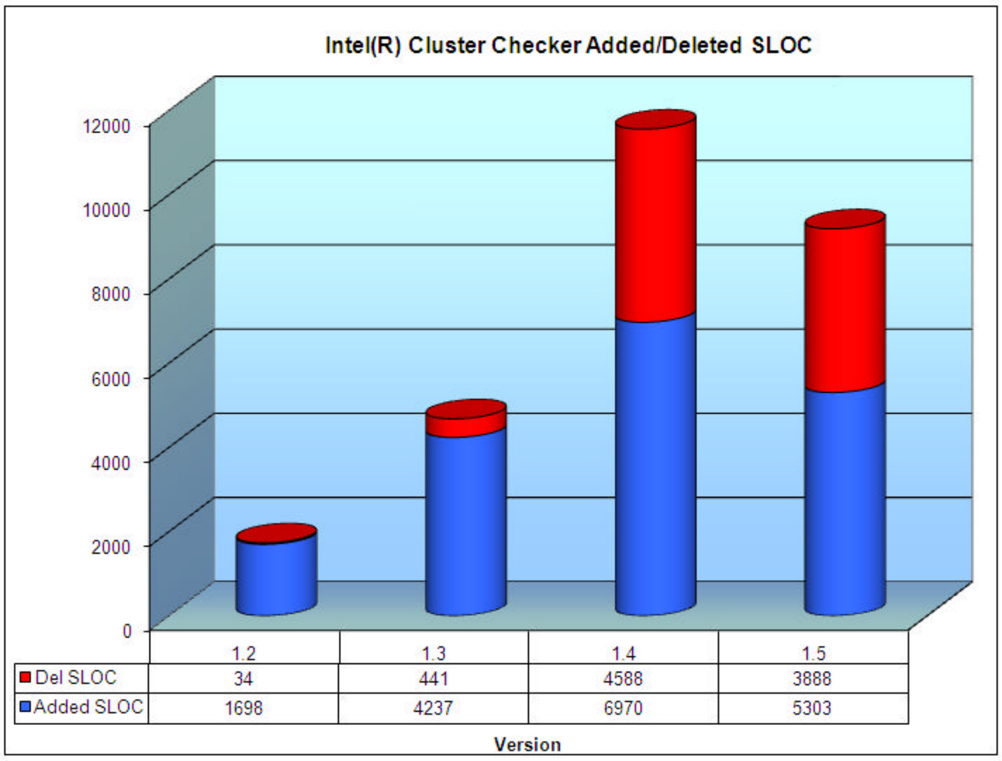
As the final result, the product team was able to release, in less time than before, higher quality versions, validating with key stakeholders on the progress and the implementation maturity for all the requested features and allowing even extra features outside the planned ones to be included. It is worth mentioning the integration of the agile methodology within the CMMi initiative at ASDC, resulting in Cluster Checker being one of the projects certified for both Level 2 and Level 3 action plans (details of this are beyond the scope of this paper). Also, very positive impact with excellent feedback was received from internal/external key stakeholders for the well-defined product development process. The practices and methodologies established in this process have been shared as *Best Known Methods* with other teams at Intel, where the added value of this experience has been the increased quality and customer satisfaction without impacting on available project resources. Smart automation and concise, but flexible, processes is a must-have for small teams and projects with a wide range of requirements. Assuring a continuous quality level through the entire development process through automation guarantees a reliable and healthy product and makes the project reach an important level of stability.



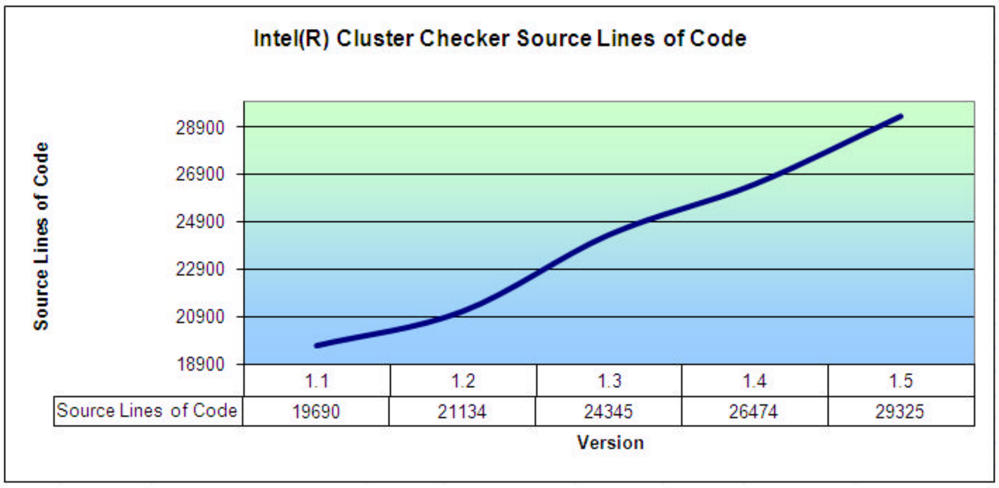
Product	Weeks	# Bugs	# Features	Issues	Issues per Week
🟨 Intel Cluster Checker 1.2	16	17	8	25	1.56
🟩 Intel Cluster Checker 1.3	14	81	16	97	6.93
Improvement	12.50%	476.47%	200.00%	388.00%	443.43%

1.2	1.3	1.4	1.5
1.1875	1.785714286	3.586206897	4.416666667

Figure 6a. Improvements achieved after process transition.

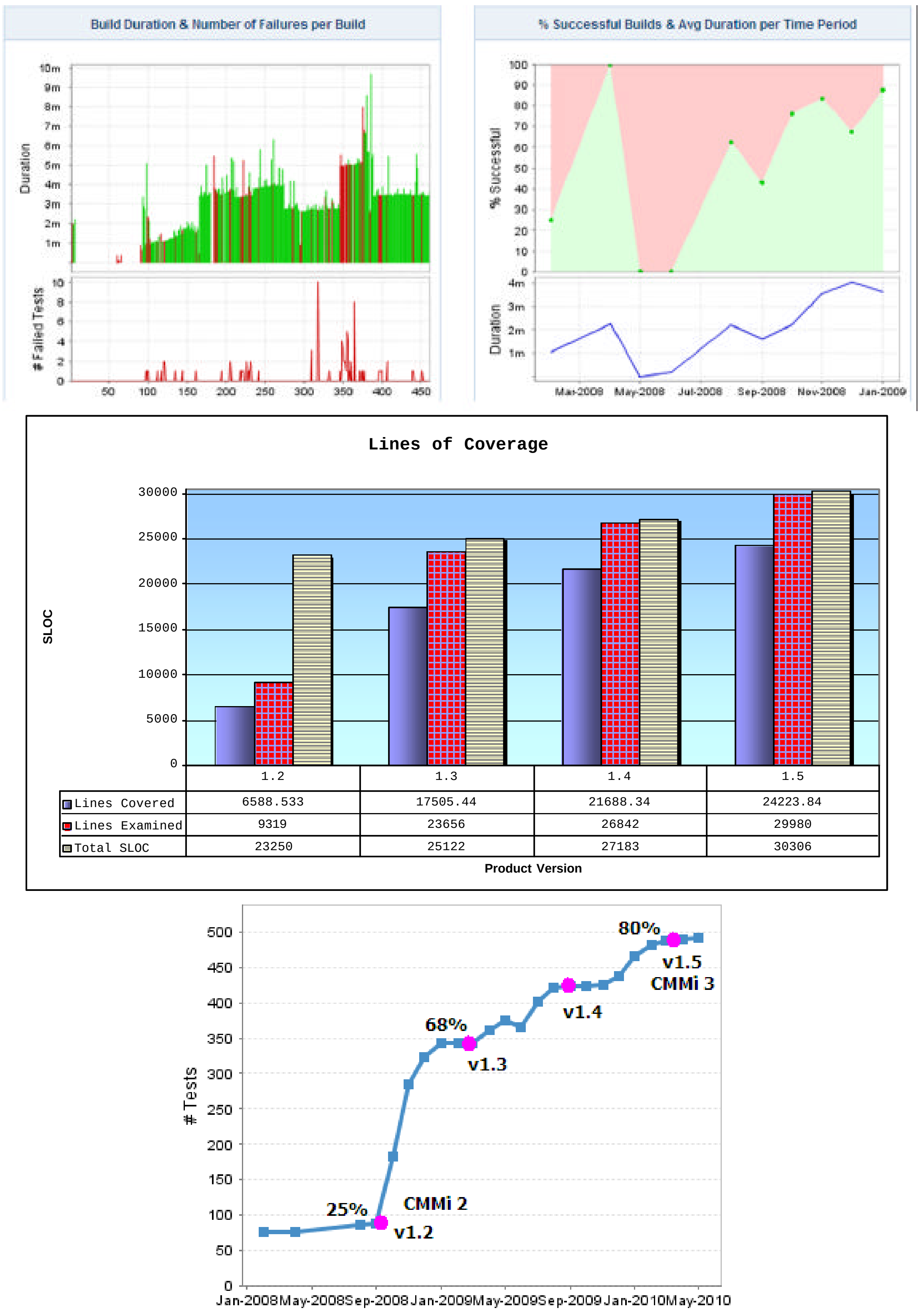


Version	1.2	1.3	1.4	1.5
🟥 Del SLOC	34	441	4588	3888
🟦 Added SLOC	1698	4237	6970	5303



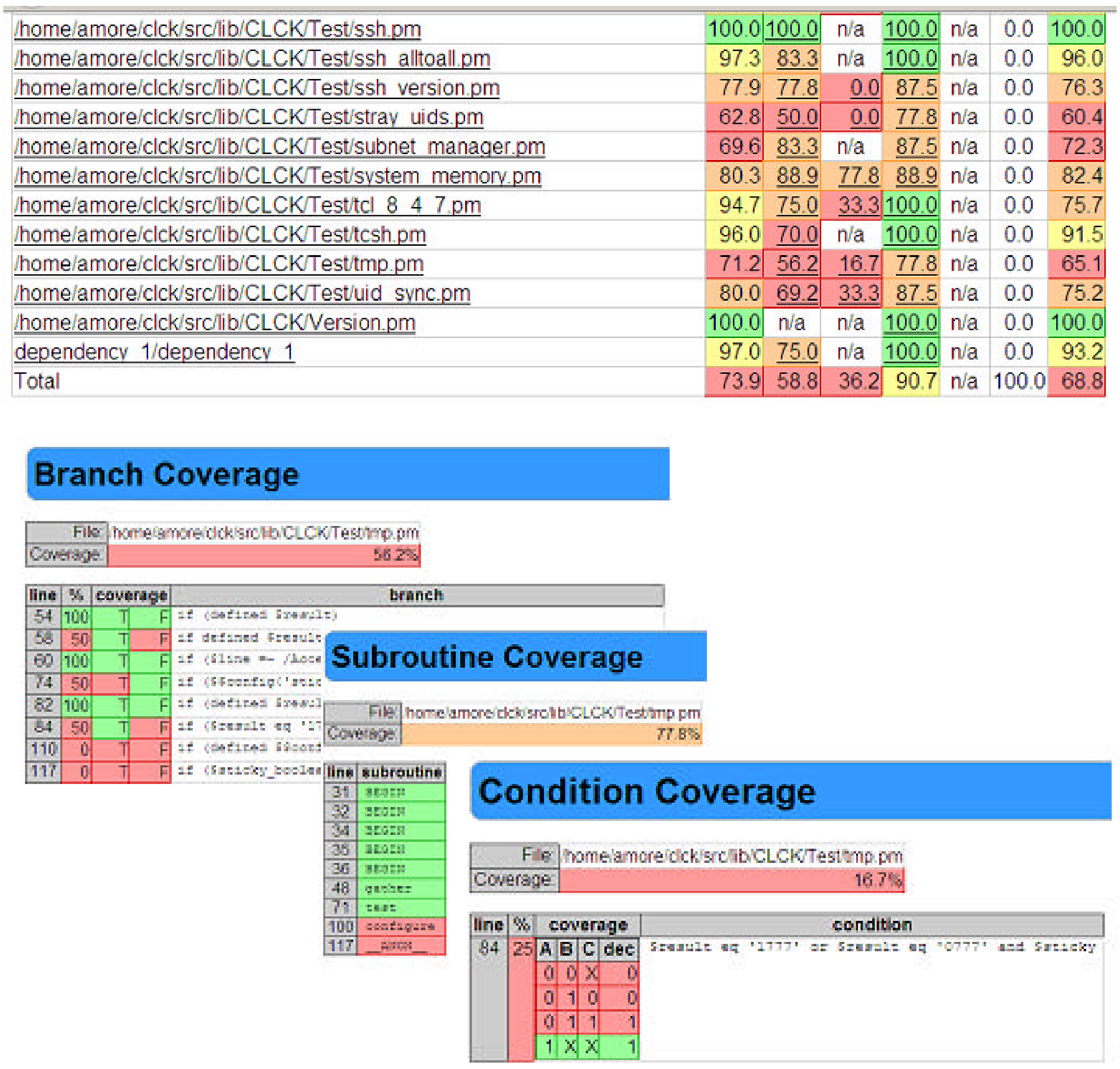
Version	1.1	1.2	1.3	1.4	1.5
Source Lines of Code	19690	21134	24345	26474	29325

Figure 6b. Improvements achieved after process transition.



	1.2	1.3	1.4	1.5
🟦 Lines Covered	6588.533	17505.44	21688.34	24223.84
🟥 Lines Examined	9319	23656	26842	29980
▤ Total SLOC	23250	25122	27183	30306

Figure 7. Growth of test cases, global coverage and build evolution.



/home/amore/clck/src/lib/CLCK/Test/ssh.pm	100.0	100.0	n/a	100.0	n/a	0.0	100.0
/home/amore/clck/src/lib/CLCK/Test/ssh_alltoall.pm	97.3	83.3	n/a	100.0	n/a	0.0	96.0
/home/amore/clck/src/lib/CLCK/Test/ssh_version.pm	77.9	77.8	0.0	87.5	n/a	0.0	76.3
/home/amore/clck/src/lib/CLCK/Test/stray_uids.pm	62.8	50.0	0.0	77.8	n/a	0.0	60.4
/home/amore/clck/src/lib/CLCK/Test/subnet_manager.pm	69.6	83.3	n/a	87.5	n/a	0.0	72.3
/home/amore/clck/src/lib/CLCK/Test/system_memory.pm	80.3	88.9	77.8	88.9	n/a	0.0	82.4
/home/amore/clck/src/lib/CLCK/Test/tcl_8_4_7.pm	94.7	75.0	33.3	100.0	n/a	0.0	75.7
/home/amore/clck/src/lib/CLCK/Test/tcsh.pm	96.0	70.0	n/a	100.0	n/a	0.0	91.5
/home/amore/clck/src/lib/CLCK/Test/tmp.pm	71.2	56.2	16.7	77.8	n/a	0.0	65.1
/home/amore/clck/src/lib/CLCK/Test/uid_sync.pm	80.0	69.2	33.3	87.5	n/a	0.0	75.2
/home/amore/clck/src/lib/CLCK/Version.pm	100.0	n/a	n/a	100.0	n/a	0.0	100.0
dependency_1/dependency_1	97.0	75.0	n/a	100.0	n/a	0.0	93.2
Total	73.9	58.8	36.2	90.7	n/a	100.0	68.8

Branch Coverage

File:	/home/amore/clck/src/lib/CLCK/Test/tmp.pm
Coverage:	56.2%

line	%	coverage		branch
54	100	T	F	if (defined $result)
58	50	T	F	if defined $result
60	100	T	F	if ($line =~ /Loca
74	50	T	F	if ($$config{'stic
82	100	T	F	if (defined $resul
84	50	T	F	if ($result eq '17
110	0	T	F	if (defined $$cond
117	0	T	F	if ($sticky_boolea

Subroutine Coverage

File:	/home/amore/clck/src/lib/CLCK/Test/tmp.pm
Coverage:	77.8%

line	subroutine
31	BEGIN
32	BEGIN
34	BEGIN
35	BEGIN
36	BEGIN
48	gather
71	test
100	configure
117	__ANON__

Condition Coverage

File:	/home/amore/clck/src/lib/CLCK/Test/tmp.pm
Coverage:	16.7%

line	%	coverage				condition
84	25	A	B	C	dec	$result eq '1777' or $result eq '0777' and $sticky
		0	0	X	0	
		0	1	0	0	
		0	1	1	1	
		1	X	X	1	

Figure 8. Assisted coverage extraction and gaps identification.

References

1. http://www.intel.com/go/cluster

2. http://www.top500.org

3. http://sas.sei.cmu.edu/pars/

4. Cohen, D., Lindvall, M., Costa, P.: An introduction to agile methods. In Advances in Computers. 62, 2-67 (2004).

5. Joseph, E.: HPC cluster market trends. Tech. rep., IDC Corp. (May 2007).

6. Joseph, E., Conway, S., Walsh, R., Wu, J., Lee, D.: Worldwide technical computing server 2008 top 10 predictions. Tech. rep., IDC Corp. (January 2008).

7. Mugler, J., Naughton, T., Scott, S., Barrett, B., Lumsdaine, A., Squyres, J., des Ligneris, B., Giraldeau, F., Leangsuksun, C.: OSCAR clusters. In: Linux Symposium (June 2003).

8. Papadopoulos, P., Katz, M., Bruno, G.: NPACI rocks: Tools and techniques for easily deploying manageable linux clusters. In Concurrence and Computation: Practice and Experience. 15(7,8), 707-725 (2003).

9. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press (2004).