Application Agility in Action on z/TPF

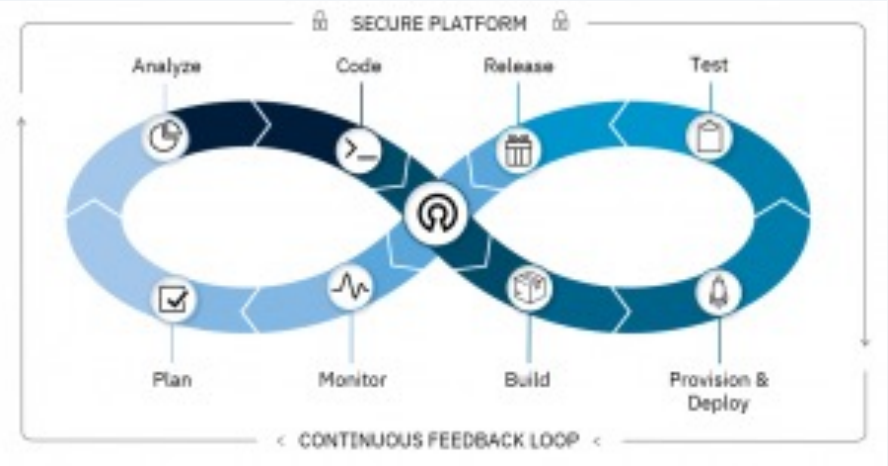
Josh Wisniewski
Dan Gritter
Matt Gritter
Kishore Nagareddy

IBM Z

IBM

Contents

Introduction



Application modernization is a strategic initiative for many z/TPF customers and IBM.

A cornerstone of many of these initiatives are DevOps and Agile practices.

It is critical to have the right tools and frameworks in place to facilitate DevOps and Agile practices.

Modern application development trends toward an iterative cycle of changes. The goal of this type of development is to continually improve the changes being made to an application with each pass through the cycle.

This presentation will walk through a variety of z/TPF tools that are **available today** that can make an application developer's job easier to accomplish, realize shift-left development savings, and help them produce better quality code.

Introduction

Andrew
New Hire
Application Developer

Andrew is a new hire and is new to z/TPF development.

His department embraces a variety of DevOps practices including the following practices specifically for developer activities:

- Test Driven Development
- Continuous testing
- Continuous monitoring

Andrew has been given the task to modify an existing application.

Scope of work

Andrew breaks the development work down into the following pieces:

- The most resource intensive core functionality.
- Edge and error cases.
- Bells and whistles.

Andrew knows that his code changes must not use an excessive amount of resources on the z/TPF system and so he first pilots the resource intensive core functionality.

z/TPF automated test framework

Andrew's department puts a large amount of emphasis on automated testing.

As such, the first thing Andrew does is to leverage the z/TPF automated test framework.

He codes some test cases to drive the most resource intensive core functionality, if they do not already exist.

Create a test case

```
#include <tpf/c_devops.h>

TPF_TESTSUITE("ibm.common.deployment","CFD*")

TPF_TESTCASE (comDep_009, "Verify no conflict if
phase completes“) {

TPF_TC_ERROR("Expected AAAA0033E message not
displayed");


}
```

The z/TPF automated test framework provides a coding framework that allows you to quickly create test cases that verify successful results, anticipated error behaviors, etc.

The z/TPF automated test framework provides the ability to quickly define test suites, test cases, error messaging, and much more.

Running a test case

Andrew commits his test case code.

The DevOps system automatically:

- Builds the code.
- If there are no build errors, deploys the code to his test system.
- Executes the test cases.
- Notifies Andrew of the success or failure of his test cases.

Test case results

```
CSMP0097I 17.10.27 CPU-B SS-BSS  SSU-HPN  IS-01

DEVO0004I 17.10.27 PROCESSING FOR THE SELECTED TEST
CASES IS STARTED.+

CSMP0097I 17.10.29 CPU-B SS-BSS  SSU-HPN  IS-01

ERROR IN QCDP,qcdp.c:1195

  aaaa_find returned incorrect data _

TEST CASE COMPLETED IN 1308ms - FAILED - comDep_014

END OF DISPLAY+

CSMP0097I 17.10.29 CPU-B SS-BSS  SSU-HPN  IS-01

DEVO0020E 17.10.29 1 TEST WERE COMPLETED.

                  0 PASSED, 1 FAILED, 0 SKIPPED+
```

In accordance with Test Driven Development practices, Andrew ensures that his new test cases fail. He has not made the application changes yet and so ensuring his test cases fail ensures that they are properly validating the results.

Also, the ZDEVO command is available to allow you to run test suites, run individual test cases, see summaries of success/failures and much more.

Code the pilot and leverage the test cases

Now that Andrew has the test automation in place, he can pilot the most resource intensive core functionality.

As he makes his code changes, he continuously leverages the DevOps system to run the automated regression tests so that he can discover as early as possible when a code change has broken existing function. This enables him to fix these breakages more quickly.

He completes the pilot effort and the application has been modified at its core.

Collecting metrics

Next, Andrew needs to ensure that the code changes he's made for the most resource intensive core functionality do not use more resources on the system than anticipated.

He uses real-time runtime metrics collection to determine the usage of key resource metrics.

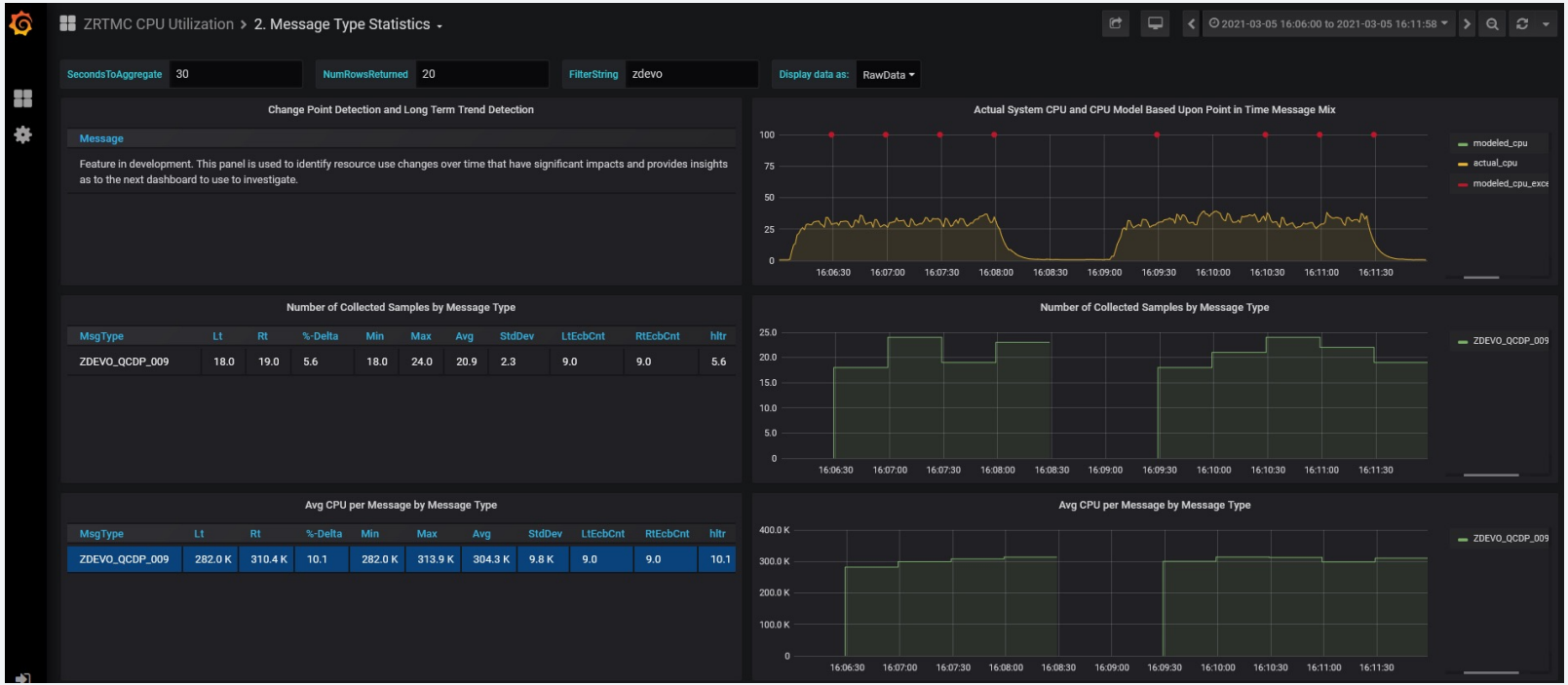
First, he uses the automated cases to capture a baseline runtime metrics collection result.

Then he uses the automated cases to capture runtime metrics collection results against his updated code.

If needed, he can run the test automation repeatedly to capture averages.

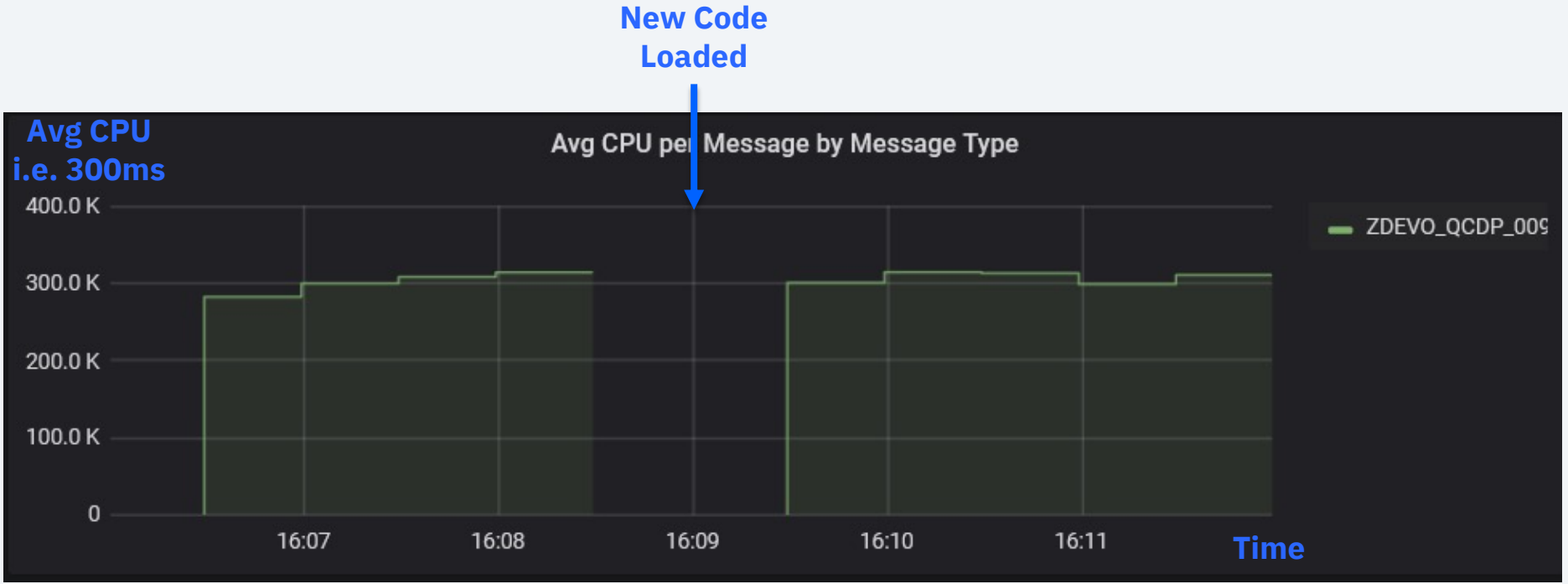
Real-time runtime metrics collection results

Andrew opens ZRTMC CPU Utilization > Message Type Statistics dashboard in Grafana in his web browser.



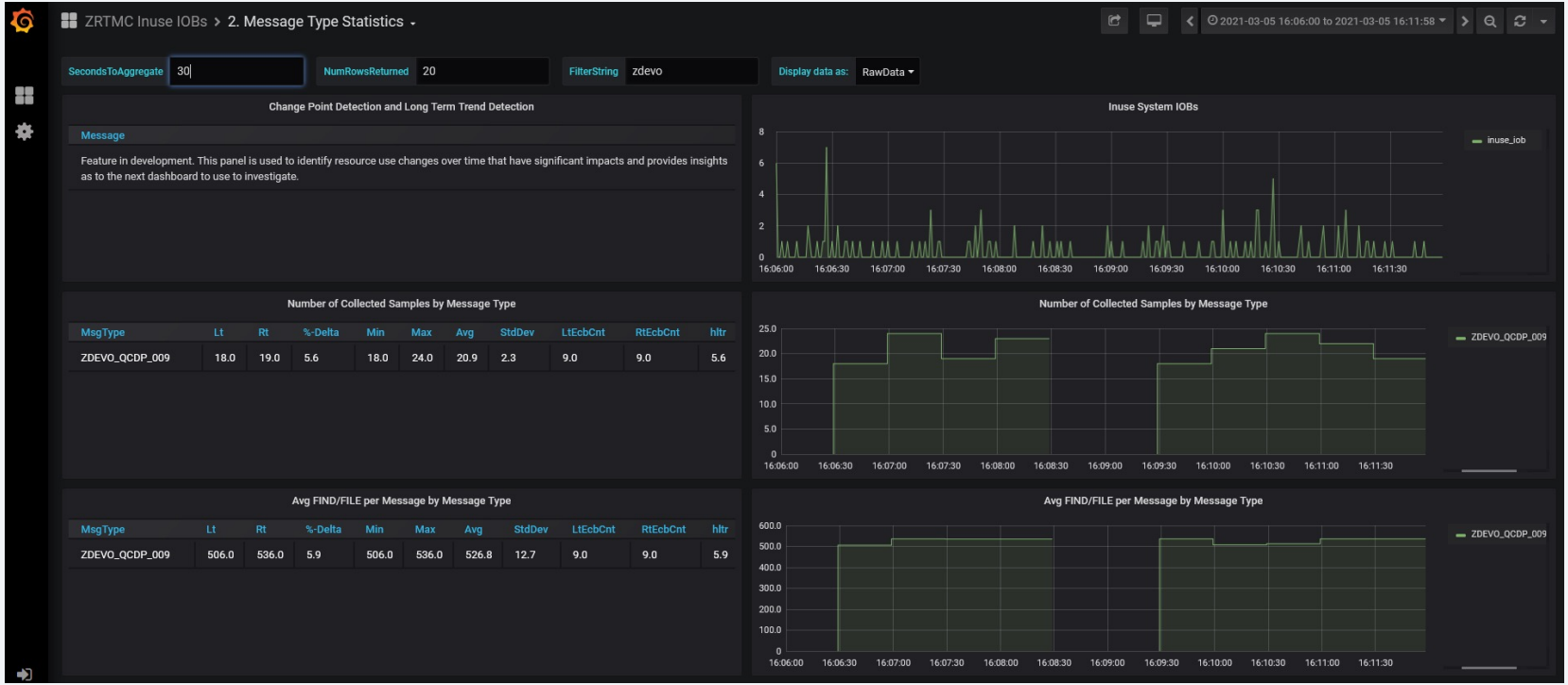
Real-time runtime metrics collection results

He can see that CPU used by the baseline and with his application changes is roughly unchanged.



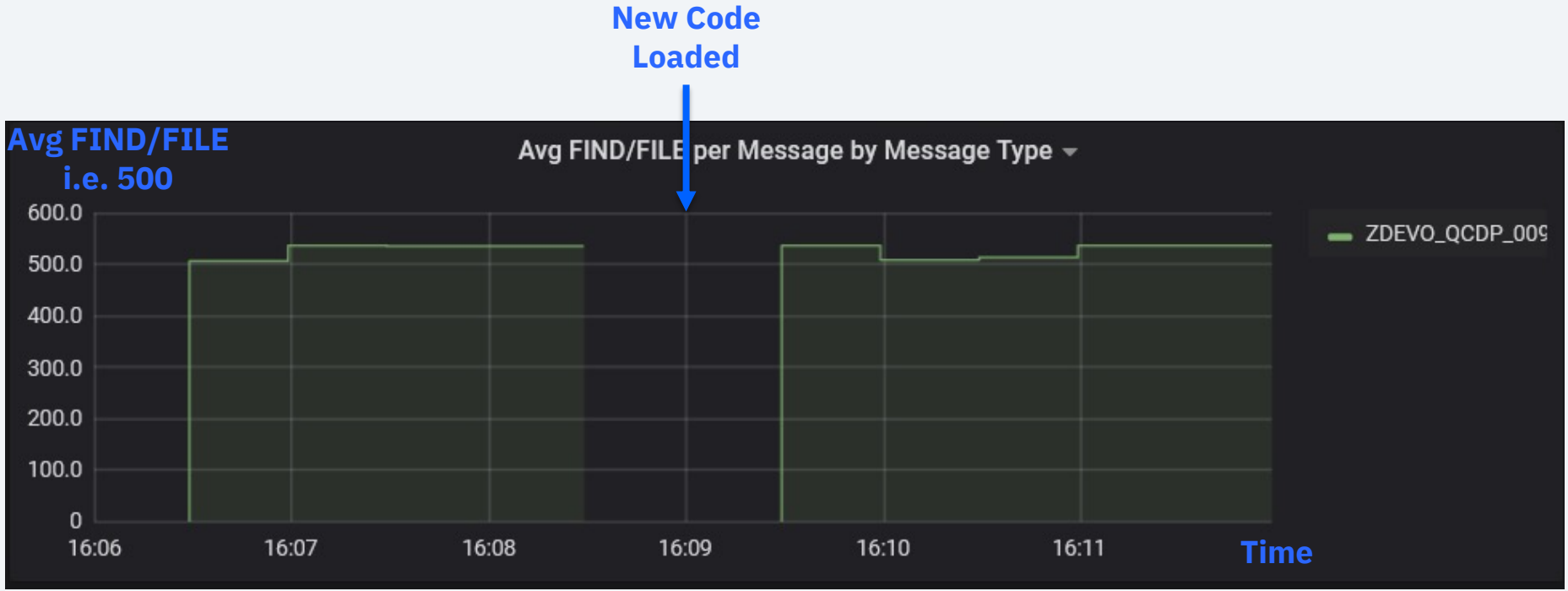
Real-time runtime metrics collection results

Andrew opens ZRTMC Inuse IOBs > Message Type Statistics dashboard in Grafana in his web browser.



Real-time runtime metrics collection results

He can see that the FIND/FILE used by the baseline and with his application changes is roughly unchanged.



Verify metric collection results

If the resource usage of his code was not acceptable, he could address the code or design issues before having invested a ton of time and work into a particular implementation.

The result is shift left savings by catching the issues early in development as opposed to late in testing or in the field.

Andrew finds that the resource usage of his code changes is within his acceptance criteria specified by the application architects.

Next steps

Andrew can now move on to:

- Formalizing the most resource intensive core functionality.
- Edge and error cases.
- Bells and whistles.

Leveraging the workflow

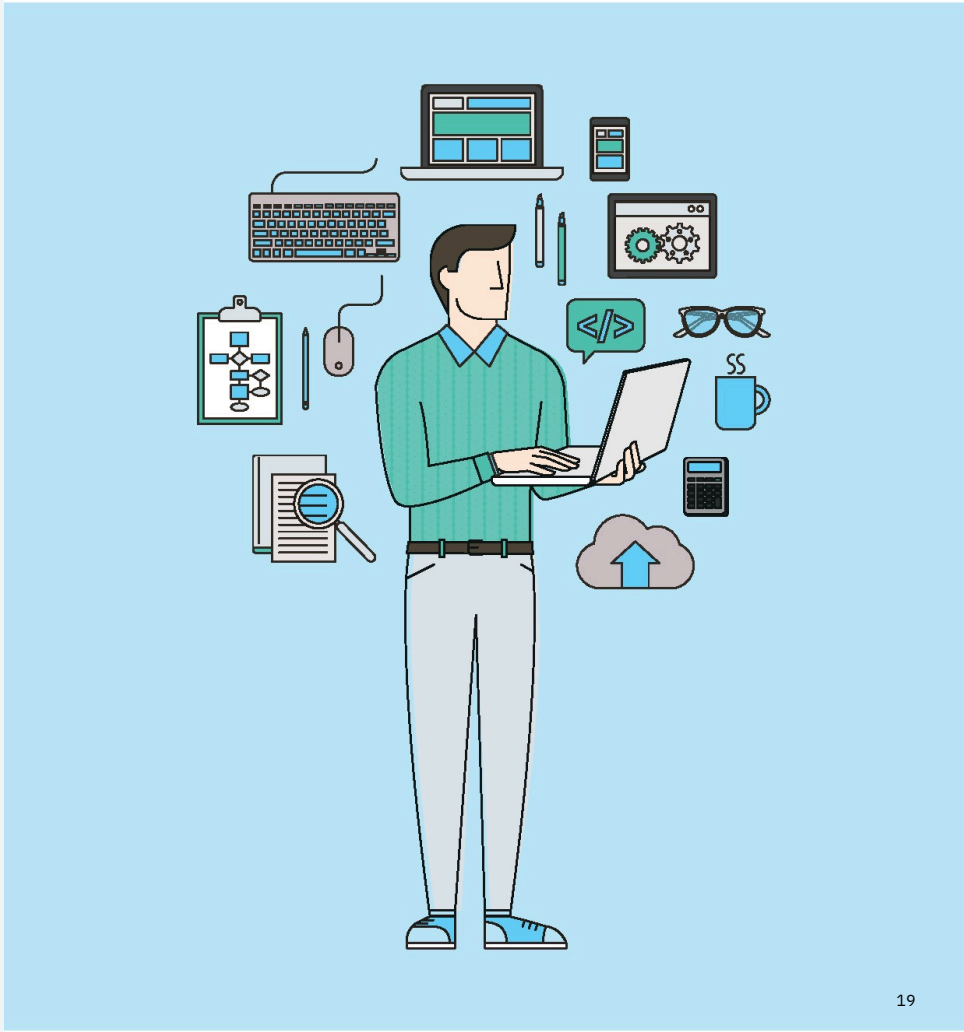
As previously described, Andrew:

- Codes the z/TPF automated test framework test cases with validation to spec. If the callee does not exist, it is stubbed out. Test cases fail.
- Codes the functionality. As functionality is put in place, fewer and fewer test cases fail.
- Leverages the DevOps system to continuously run the automated testing to see the functionality is coded correctly and that he hasn't broken yesterday's efforts in the process.
- Continuously runs real-time runtime metrics collection to verify that resource usage remains unchanged from previous baselines. That is, he hasn't accidentally introduced a resource usage issue while implementing additional functionality.

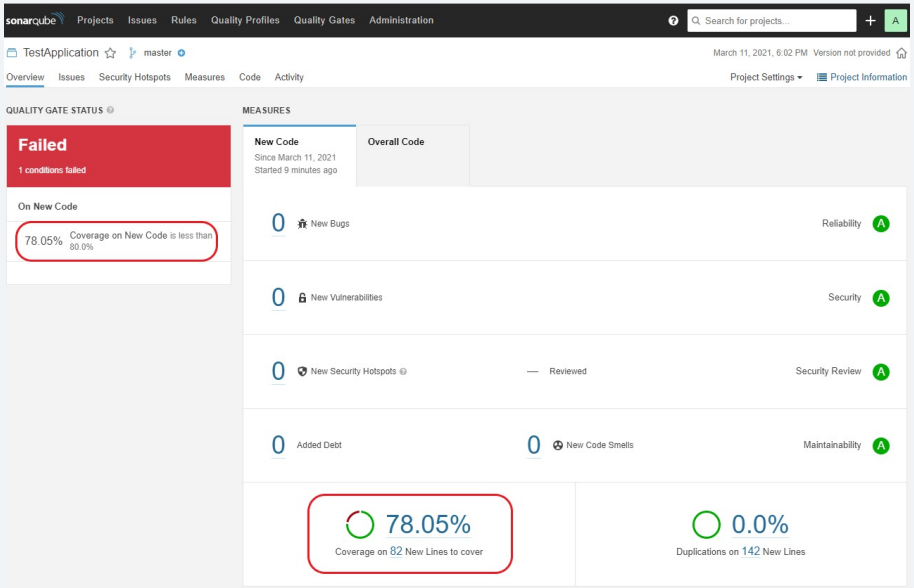
Code coverage

As Andrew approaches the end of his coding and unit test phase, he needs to ensure that his test cases are adequately testing new and changed code.

Andrew leverages z/TPF code coverage and SonarQube to ensure that all new and changed lines of code have been tested by his automated test cases.



SonarQube - Results



SonarQube provides the capability to view code coverage information against new and changed code in a project.

The development project must be kept in a source control management system that SonarQube supports to determine what code is new or changed. We will use a temporary local Git repository as an example.

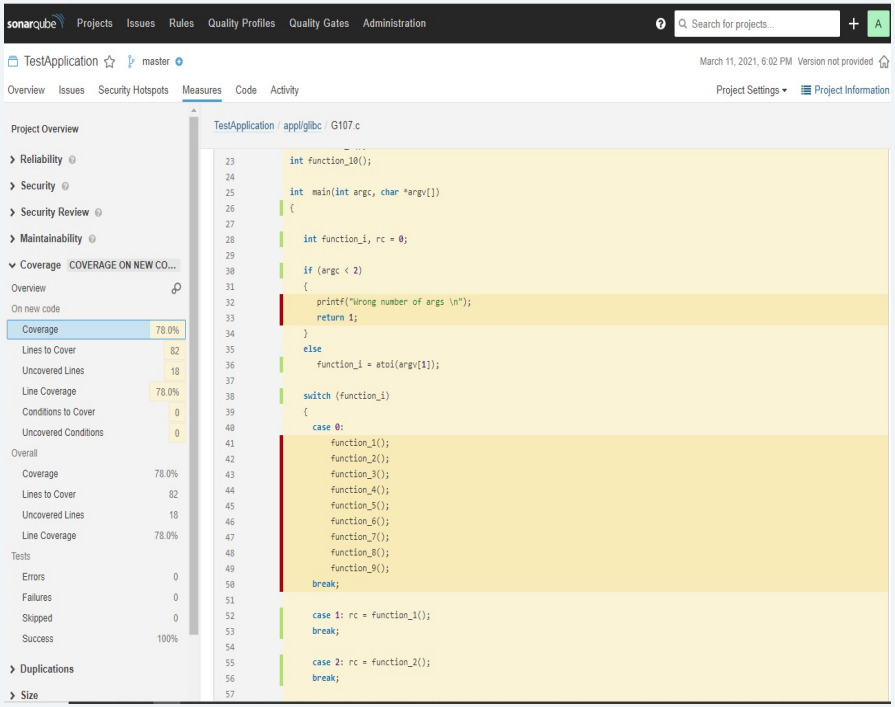
The Sonar C++ plugin provided by the Sonar Open Community must be installed on the SonarQube server. The plugin supports viewing code coverage information in the Cobertura report format.

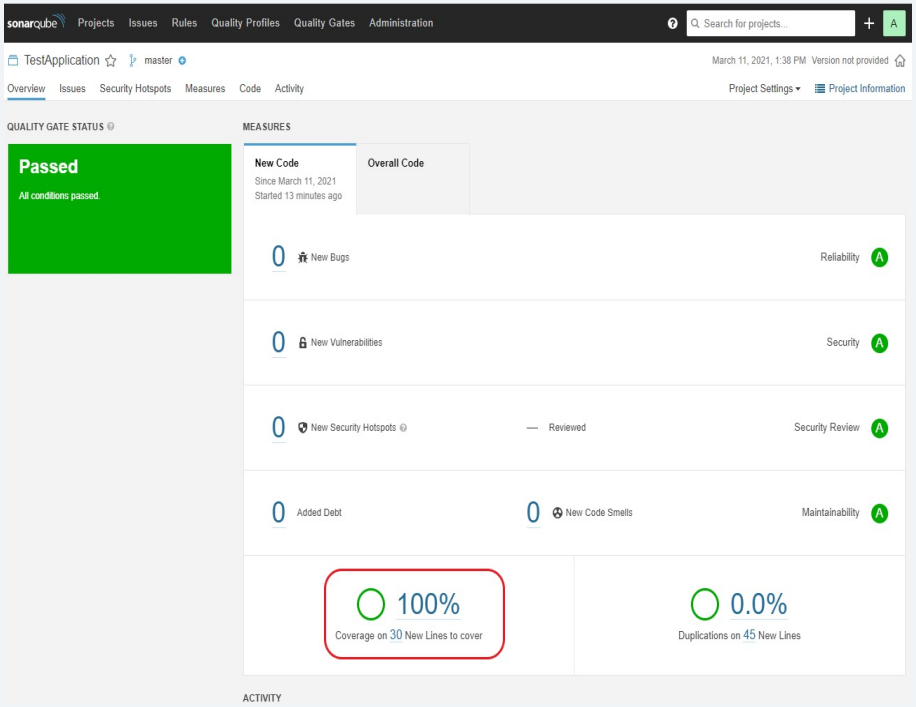
SonarQube can also store and display additional metrics for your development projects.

Andrew finds that his project failed the quality gate condition because not enough new and changed code was covered by his tests.

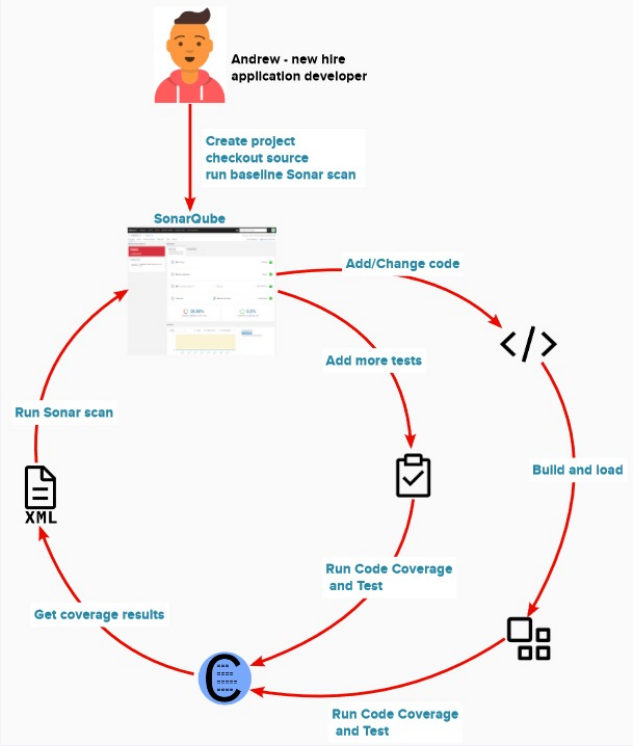
Improving code coverage

Andrew then examines his code changes to determine which lines were not covered by his test cases and adds more test cases to achieve the desired code coverage results of 100%.





Code coverage workflow



Check out files for development project

Wrap development project in a Git repository

Run initial scan to set baseline

Add / change code to development project

Build / load to test system

Start code coverage collection

Run test cases

Stop code coverage collection

Retrieve code coverage report

Scan project

Check SonarQube dashboard

Add/change test cases to achieve desired coverage

Repeat as desired

Code coverage example scripts

IBM has provided sample scripts to run scans on the project and to invoke the code coverage REST services on the z/TPF system.

The scripts can be found on the drivers download page:

https://www.ibm.com/support/pages/tpf-family-products-drivers-ztpf-11-ztpfdf-11

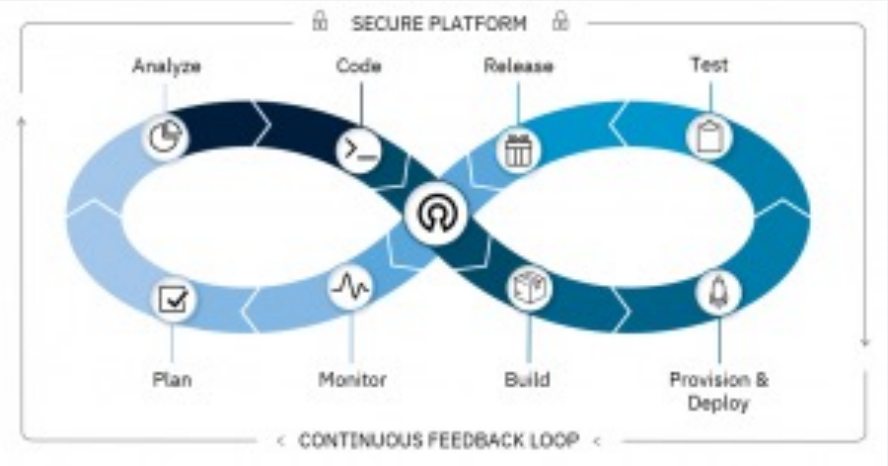
Check the blog post: Code coverage reports as a service on z/TPF

Development complete

Andrew completes his efforts on time, to specification, with test automation, with confidence that all paths are adequately tested and without introducing resource usage issues to the system.

Andrew can now move his project through the enterprise test phases with confidence.

Final thoughts on our story…

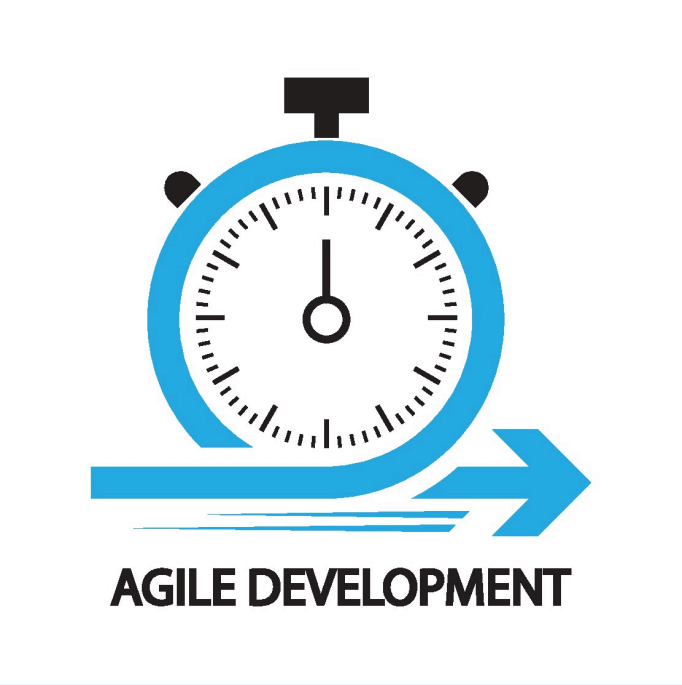


Leveraging the z/TPF automated test framework provides tremendous value by reducing effort and improving quality. Notice that in our application development story, the automated test cases were a reusable asset used to:

- Ensure newly written code correctly satisfied acceptance criteria.
- Ensure previously delivered code was not accidentally broken.
- Validate system resource usage was inline with the specification using real-time runtime metrics collection.
- Verify that all new and changed code had been tested satisfactorily with code coverage.

Further, these test artifacts can be used by future regression test and other efforts to make sure that the delivered functionality continues to work as designed.

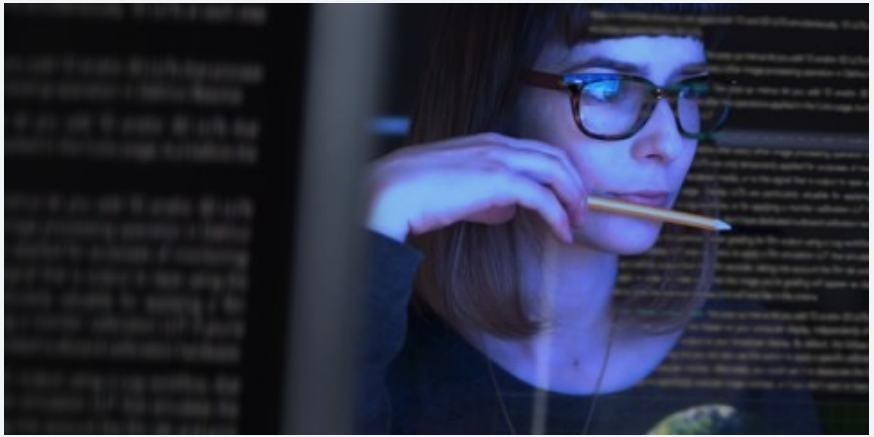
Conclusion



Using the z/TPF automated test framework, real-time runtime metrics collection and code coverage with support for new and changed code as part of an iterative application development cycle, you can:

- Quickly identify coding mistakes and design issues while development occurs.
- Take advantage of shift left savings by identifying resource usage issues early in the development cycle.
- Be confident that application changes are fully tested and of the highest-quality.

APAR Numbers



- z/TPF automated test framework:
  - PJ43782 – 08/2018 – Initial support.
  - PJ45488 – 12/2018 – REST support.
  - PJ45801 – 08/2019 – Allow users to override or intercept functions.
- Real-time runtime metrics collection:
  - PJ45657 – 05/2020 – Initial support.
  - PJ46295 – 11/2020 – Improve starter kit.
  - z/TPF real-time insights dashboard starter kit download page.
- Code coverage updates:
  - PJ45090 – 03/2018 – Scriptable code coverage.
  - PJ45949 – 12/2019 – SonarQube support.
  - PJ46334 – 02/2021 – Cobertura format.

Call to action

Application modernization is a strategic initiative for IBM.

This includes facilitating our customers on their DevOps and Agile journeys.

We ask that you:

- Leverage the tools we've made available and let us know how they can be improved for your needs and practices.
- Work with us to fill the gaps in your tooling, practices and processes.
- Help us build the future of the z/TPF development and operations environment.

We would love to have further design discussions on these topics with you: danielle.tavella@ibm.com

Teaser for z/TPF message analysis tool presentation

Andrew
New Hire
Application Developer

Remember back in our story when Andrew used real-time runtime metrics collection to determine **if** a resource usage issue existed? He found “that the resource usage of his code changes was within his acceptance criteria specified by the application architects.”

What if he had introduced a resource usage issue? How would he determine **where** the resource usage issue was in the application code?

In the near future, he could use the z/TPF message analysis tool to determine where the resources were being used in the application. We’ll discuss this new tool further in a separate presentation in the Operations and Coverage subcommittee next week.

Thank you

Virtual TPFUG Q&A

Question	Answer
Multiple customers discussed how they are at various stages of adopting the automated test framework. Some customers expressed interest in how they can speed up their adoption or make the framework more inviting to use.	IBM and other customers with more experience using the automated test framework discussed how once you get your feet wet, you'll find it is quick and easy to use. You can also create code templates in your IDE to make creating test cases more quickly. IBM also discussed the potential of further enhancing the learning platform. IBM also asked one of the customers with more experience to share their experience with the community.
Different customers asked about more complex scenarios or various conditions.	IBM encouraged customers to open RFEs (request for enhancement) and to work with us through sponsor user engagements to make framework more robust.

Virtual TPFUG Q&A

Question	Answer
A customer asked for documentation on the intercept/invoke support for the automated test framework.	https://www.ibm.com/docs/en/ztpf/2021?topic=zca-tpf-tc-intercept-intercept-call-return-from-function
	Here is an additional link to help with the challenges of Test Driven Development (TDD): https://medium.com/ibm-garage/when-tdd-gets-hard-fc14136c3f44