28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Large-Scale Integration Testing at Microsoft

Jean Hartmann

Test Architect, Developer Division Engineering Systems

Microsoft Corp.

Redmond, WA 98052

Tel: 425 705 9062

Email: jeanhar@microsoft.com

Abstract

Given the large number of components and dependencies, substantial time and resources are expended by Developer Division teams in integrating and validating these components into the product. In past releases, this flow of code between product branches was often hampered by component teams inadequately qualifying their code contributions in these branches during reverse integrations (RIs) and compounded by the fact that important dependencies were being taken on their code, often without their knowledge. As a result, when those dependent component teams merged those unstable code contributions into their own code base, that is, product branches as part of their forward integrations (FIs), breaking changes proliferated with respect to product and test code - important integration points had not been exercised by the RI'ing team. All teams subsequently paid a heavy price in terms of failure analysis. With this scenario repeating itself numerous times during the development cycle, dependent teams became wary of taking FIs and code velocity began to slow. This resulted in a product, which was not updated often enough and many times ended up in an unstable state.

In this paper, we describe an ongoing test improvement initiative, which supports efforts to reinvigorate this code flow, and ensures that code quality is not compromised. We more closely examine some of the past issues that led to the stagnating code flow and describe our new approach to integration testing by focusing on the new strategy, processes and tools.

Biography

Jean Hartmann is a Principal Test Architect in Microsoft's Developer Division with previous experience as Test Architect for Internet Explorer. His main responsibility includes driving the concept of software quality throughout the product development lifecycle. He spent twelve years at Siemens Corporate Research as Manager for Software Quality, having earned a Ph.D. in Computer Science in 1993, while researching the topic of selective regression test strategies.

1. Introduction

As the size and complexity of flagship products, such as Visual Studio 2010, continues to increase, the number of challenges surrounding product code integration and its validation are also rising. These challenges are technical and logistical in nature, yet typical of any large-scale software development. In our case, we have a large number of individual teams working around the world on different parts of the product in so-called *product branches*. They contribute their code components and test collateral at regular intervals via *Reverse Integrations* (RIs) to the existing code base for the entire product. Conversely, teams regularly absorb that newly integrated code via *Forward Integrations* (FIs) into their product branches. Thus, a code flow is established for the product and ultimately enables the product to be shipped with the required features.

The key challenge that we face involves *improving code velocity* with RIs and FIs occurring on a more regular and frequent basis. Whenever teams, that is, product branches are broken by flawed code changes propagating throughout the product (build tree), everyone has to expend significant resources fixing test and/or product issues. Depending upon the number and frequency of such breaking changes, teams become reluctant to take FIs and in turn, initiate fewer RIs. They, in effect, start to isolate themselves more and more in order to retain their development momentum within a branch. The result of this growing isolationism is that the rate of code flow begins to slow more and more, producing less than optimal integration results around major product milestones.

A closely related issue here is integration testing; in particular, the early detection and validation of key product integration points. If an integration test suite is not effective and efficient at validating the code being integrated as early as possible, then this can seriously exacerbate code flow problems. Our existing suite had many issues including a lack of: clear test selection criteria, formalized test case review/update processes, API-based automation, standardized logging mechanisms and test portability. Test collateral was too feature-centric, difficult to execute and debug for anyone except test owners, and often unstable. Years of growth had left the test suite with numerous ‘stale’ or redundant tests. The irony was that our integration suite was frequently executed and would pass without failures, yet product bugs, especially integration issues continued to be propagated during FI’s.

Finally, the issue of code flow and integration testing takes on a bigger dimension, when you consider the challenges of integrating products across Microsoft. Our partners, such as Windows, rely on us to provide them with high quality drops for their respective code bases. This involves defining a larger integration suite that is capable of expressing overall product quality respective Visual Studio to such internal Microsoft partner organizations.

In the remainder of this paper, we discuss our ongoing test improvements as part of the divisional “fearless FI” initiative. Section 2 focuses on the definition, implementation and execution aspects of the new integration test suite. In Section 3, we intend to deploy that new test suite to improve our existing code flow process within the division. For Section 4, we summarize the supporting test infrastructure and tooling needs that arose during development and deployment of this new suite. Finally, Section 5 hints at some of the future work that we intend to conduct in subsequent phases of this continuous improvement initiative.

2. Developing the New Integration Test Suite

2.1 Test Selection Criteria, Review and Resolution

The key objective for our test selection process was striking the correct balance between *efficiency* and *effectiveness* of the test suite. From extensive reviews of other integration test processes within Microsoft, it appeared that very few teams, if any, were achieving such a balance. Lack of clear selection criteria and more importantly, well-defined update processes resulted in integration test suites becoming bloated over time and significantly raising test costs over several product releases.

Based on these experiences, we concluded that it would be better to apply more stringent test selection criteria from the onset, possibly resulting in a minimal test suite. We would then leverage a well-defined update process to evolve the test suite alongside the product changes during the shipping cycle. Simple and clear criteria proved effective during communications with the teams. The following key criteria were set regarding *efficiency*:

- Tests execute in a two hour timeframe (excl. setup)
- Tests are reliable, consistent and easy to debug

We applied the following key criteria regarding *effectiveness*:

- Tests exercise key product integration points
- Tests reflect customer success scenarios ('happy paths')

As suitable test cases started to be identified, we categorized the resulting tests based on the number and uniqueness of the product integration points being exercised. The result was a primary and secondary set of integration tests, collectively known as the Reverse Integration Tests (RITs). These suites contained mostly functional, but also a few non-functional, namely performance tests; the latter needed to meet the above stated efficiency criteria.

The primary test set, or (Developer) Division RITs (DDRITs), focused on validating the major integration points that coincided with the top customer success scenarios. If these integration points were broken by any RI'ing team, then the product itself would also effectively be broken upon integration. This set of tests needed to be run as part of every RI and if any test from this set failed, it would block the integration of that team's code into the main build. Any resulting bugs would be flagged as Priority 0 (highest priority) and need to be resolved within a few hours. This set of tests is currently of the order of tens of test cases to minimize execution and failure analysis time.

The secondary set of tests, or Product Unit-specific RITs (PURITs), focused on validating more team-specific, unique integration points that aligned with the remaining customer success scenarios. This set of tests would only be run by the team whom other teams took unique dependencies on. If any integration points were broken by the RI'ing team, then only the dependent teams would be affected; not the whole division. General code velocity would be maintained. Thus, any tests failing from this suite do not block integration for that team's code.

Having said that, it is in the interest of dependent/partner teams to resolve the filed bugs (Priority 1) in a timely manner - any unresolved bugs and subsequent RI of code could lead to the dependent teams taking breaking code changes via an FI, which is to be avoided. In practice, teams selected those tests

that resulted in frequent breaking changes when their dependent teams churned code, that is, public APIs. This set of tests represents tens of test cases and will be run by the various teams across product branches.

The next step after identifying suitable tests involved the review and documentation of those proposed tests. In the case of the DDRITs, a *divisional* review committee was established comprising of both development and test managers as well as senior technical staff. Their role was to review and approve proposed tests and ensure strict adherence to the given selection criteria. While test case status was tracked manually, future support is being implemented using our work item tracking in the Team Foundation Server (TFS) product.

For the PURITs, the review committee comprised of staff from the partnering, dependent teams only. For those reviews, test purpose and content were captured via Service Level Agreements or SLAs between dependent teams. Templates were provided to the teams to ensure a general level of content consistency and then stored in a central Sharepoint server location to enable easy access for viewing and updating during the shipping cycle. Pertinent information included contact information such as test owners and delegates, list of all test consumers, anticipated test execution timeframes, links to test case descriptions and IDs, etc. Going forward, these divisional and team-specific review committees will become an integral part of our new integration testing process and are critical in helping teams define new RITs, update existing ones and deprecating old tests, whenever applicable. It will support our goals of maintaining an efficient and effective test suite.

While the above test sets may be sufficient for ensuring high-quality code flow within Developer division, they are insufficient when considering integration testing *in-the-large* across divisions. With this scenario, large portions of our code base need to be integrated into another division's product on a frequent cadence and have to be of very high quality. In response to these requirements, the test selection criteria were expanded to encompass the above types of integration tests as well as additional feature (functional), performance and stress tests that thoroughly exercise those code portions being integrated elsewhere. We also relaxed the execution constraints due to the larger size of this suite; enabling a runtime of about eight hours excluding setup and installation. However, we did maintain the test case reliability and debug goals – tests from either suite need to be stable and easy to debug! Unlike the RITs, which would typically be run against a product branch whenever a team wanted to RI (aka on demand), this suite would be run against those portions of code to be integrated elsewhere on a daily basis.

Given the different test suites and number of potential issues resulting from test failures, we aim to introduce the role of *Product Unit (PU) RI* representative as part of the process. In essence, each team will have such a contact. This person will be the first line of defense, should any integration tests fail; whether for product or test reasons. That person will be responsible for initial triaging and assignment of resulting bugs, collaborating and liaising with dependent partner teams to resolve, for example, any RIT issues, updating of SLA agreements, etc. From extensive reviews of other integration test processes within Microsoft, it appeared that those divisions that introduced such positions minimized the randomizing impact of incoming bugs on the teams, that is, developers and testers. We anticipate that these contacts will also be able to liaise with test infrastructure owners, if the bugs are tooling-related issues.

2.2 Test Implementation and Purification

With appropriate initial sets of RITs reviewed and approved, the next stage of our initiative focused on effectively implementing and 'purifying' those tests. We wanted to avoid the problems we had with our

previous integration suite and meet the criteria stated above concerning test execution *efficiency* – minimizing execution time, maximizing reliability, ease of failure analysis and debugging.

The vast majority of scenario-based test cases developed at Microsoft leverage UI-based test automation. This overexposure to a 'top-down' black-box testing approach has been beset by issues associated with performance, stability and reliability. In order to avoid such issues from the onset, we mandated that the primary set of integration tests, the DDRITs, must be implemented as *API-based* scenario tests. A new API-based test automation framework was developed and deployed to support this effort. Performance improvements were significant, often reducing execution time from several minutes to seconds; if the Visual Studio start-up time is not considered. It is worth noting that as tests utilized this framework, we also needed to make product changes for access to the appropriate APIs and improve testability.

In certain cases, however, we had to fall back on tests being implemented using a 'mixed mode' – containing both API- and UI-based automation, whenever the cost of refactoring product APIs to make them testable was prohibitive. With regards to the larger number of PURITs, we could only recommend that any new tests be implemented using API-based automation; essentially we leveraged existing test collateral due to time constraints.

The other important aspect of this work related to standardizing the logging data being generated by tests to ease the failure analysis for developers and testers as well as aid debugging by any team that breaks a RIT. For this, we defined a new logging API that is being integrated into our underlying API-based test framework and could easily be shimmed into an existing test environment for any PURITs. In future, we will be leveraging and extending this API as we explore smarter failure analysis techniques.

Once implemented, all test cases were then subject to 'purification'. This meant that tests were repeatedly executed against a predetermined matrix of OS, Visual Studio SKU configurations and languages. In each case, the test execution system would monitor whether test failures occurred during execution. Tests failing purification could be repeatedly submitted until they ran flawlessly. Future versions of the purification system and process will be implementing 'test pollution' reduction measures that ensure tests leave their execution environment pristine. In future, we aim to also develop static analysis checks that enforce test design practices for UI- and API-based test development. These automated checks would run prior to test execution.

2.3 Test Execution and Reporting

Test execution and reporting leveraged the results of two related, divisional test initiatives. The first effort focused on consolidating our existing, distributed testing lab resources by moving them into a new cloud infrastructure at a geographically remote site. There, a large number of virtual machines (VMs) became available, which teams could use for testing purposes based on virtual machine images (VHDs). In addition, this effort included enhancements to our existing test case management system to enable execution of the functional RITs in the cloud.

In the past, a large percentage of time, prior to test execution, was spent with basic machine configuration tasks including the setup and installation of OS/SKU/language configurations. With the advent of clouds, such setup and installation work is minimized. A priori, VHDs are defined containing all preconfigured OS/SKU/languages configurations necessary for the test runs. These VHDs are stored as part of the cloud infrastructure. Each test run can then quickly image the VMs, install the required Visual Studio and related products and begin test execution.

Upon test failure, whether product- or test-related, the VM executing the erroneous test is saved for the developer or tester who can then remotely debug the issue. Once the issue is resolved, the VM is released back to the pool of machines.

The second effort being leveraged for integration test purposes is focusing on consolidating our heterogeneous and distributed reporting mechanisms via a central 'dashboard'. To increase transparency, that is, provide quicker and easier access to detailed integration testing results, a new reporting infrastructure is being developed that will enable the entire management chain to consume and respond to this data. This is especially important in the case of the RITs, whose execution results need to be quickly scrutinized for failure in order to respond in a short time period.

In summary, the new integration test suite can be executed by our central engineering team (DES) on behalf of our RI'ing product teams with results being reported and available for all to view in one location.

3. Deploying the New Integration Test Suite

The above sections described how we defined, implemented and executed the integration test suite. We also introduced the different stakeholders and their roles/responsibilities - central engineering team (DES), divisional and team-specific review committees, PU RI Rep and PU Developer and Test Owner. In this section, we describe how we intend to deploy the new test suite as part of our improved integration test process.

3.1 Deployment Goals

We set ourselves the following deployment goals:

- **Maintain test suite efficiency** – the performance and reliability of the tests needs to be maintained under all circumstances; maintain rapid code flow is paramount.

- **Evaluate fault detection capability** – the effectiveness of the new integration test suite should be at least on par with the old one; the most important integration bugs should be caught by both suites.

- **Evaluate test suite effectiveness and adequacy** – the new integration test suite should continue to exercise key integration points in the product as well as reflect customer success scenarios as the product evolves, that is, code churns.

3.2 Workflow

The workflow and process is currently being defined and is subject to change. Having said that we have defined the two key scenarios that are being fleshed out:

a) A team is ready to integrate or RI their code contributions
b) Product changes require the test suite to be updated

3.2.1 Preparing for Reverse Integration

Figure 1 depicts the workflow throughout reverse integration of a given team's code contributions. With the latest daily product branch (PU) build available from the build team, the team wanting to reverse integrate their latest code changes requests an integration test run from DES Test, the central engineering test team. They, in turn, schedule a run of all applicable integration tests in the cloud. If all tests pass successfully, that is, the reporting dashboard lights up green, that team's code contributions can be submitted and merged into the main product code base. Consuming teams should in theory be able to take a fearless FI and not be broken.

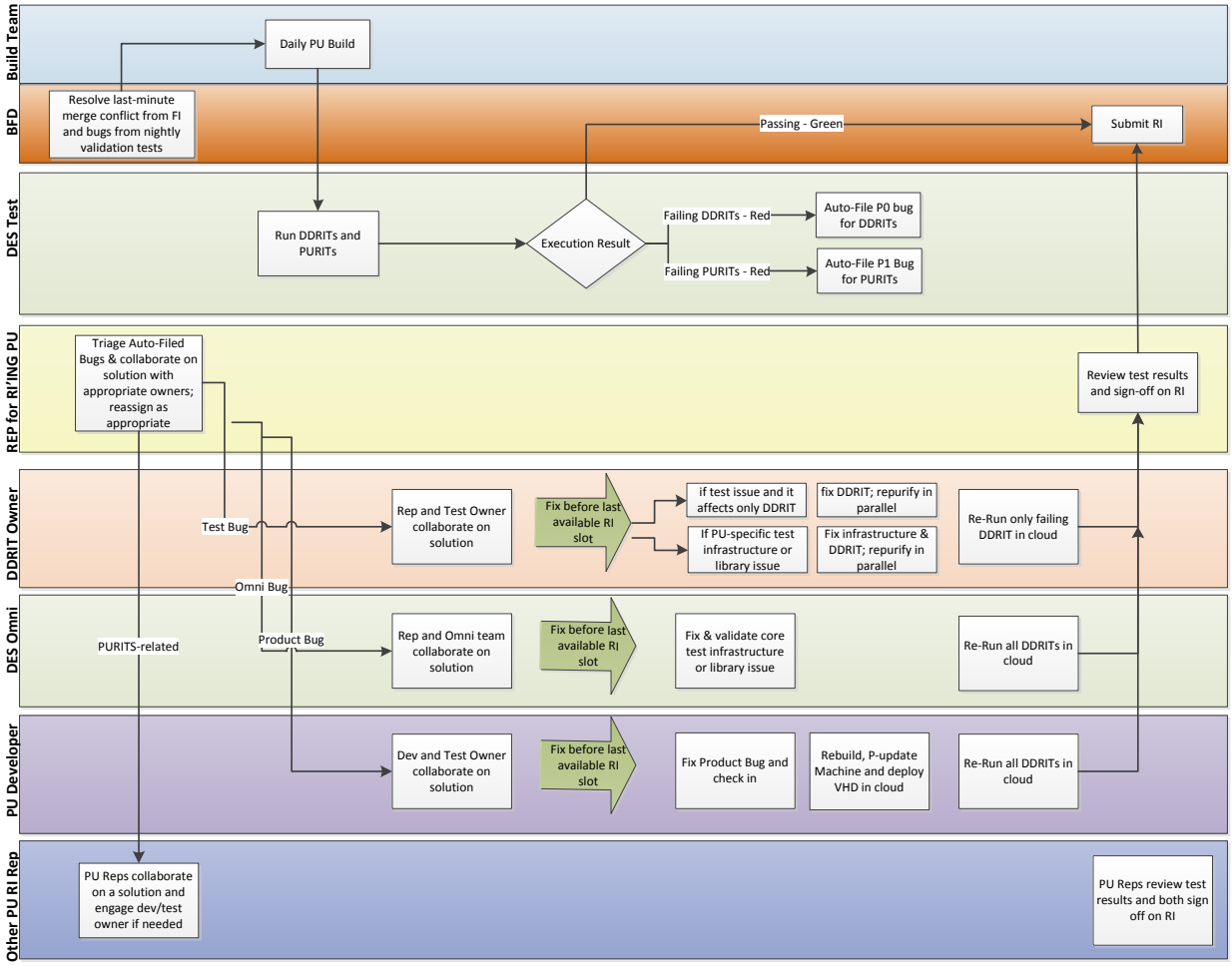

Figure 1: Integration Testing Process during RI

Otherwise, the reporting dashboard would light up red. Bugs are automatically filed with status appropriate to the test case – P0 bugs for DDRITS and P1 bugs for PURITS. The representative for the RI'ing PU conducts the initial triage of the incoming, auto-filed bugs for DDRITs and reassigns them as follows:

- **Product bugs** are filed against developers
- **Test bugs** are filed against test owners
- **Test infrastructure bugs** are filed against Omni team (centralized engineering team)

The representative also initiates discussions with other product teams, if the failure is PURITs-related. With these test failures, the workflow has been prescribed and agreed upon in the service level agreement (SLA). It follows a pattern similar to the one described in the next few paragraphs. If bugs are resolved in an appropriate and timely manner, the representative will review and close any outstanding bugs as well as sign-off on the RI.

For developers, this process implies that product fixes are necessary to the PU build. Once completed, the *entire suite of applicable RIT tests* needs to be rerun to ensure no regressions. To achieve this, the PU build must be updated and a new VHD created for rerun in the cloud. Developer and PU RI rep must sign off on the work upon completion.

For test owners, this process requires fixing of the test bug and revalidating the *affected RIT test case* to ensure no regressions. Re-purification is also necessary to maintain test suite efficiency, but does not block sign-off. Test owner and PU RI rep must sign off on the work upon completion.

For the DES Omni team, this process involves fixes to our core test frameworks and/or libraries. As these frameworks impact all RITs tests and are critical to the success of every team's RI, this central engineering developer team needs to rerun the *entire suite of applicable RIT tests* to ensure no regressions.

Up to 5 RI slots are available per day for teams to RI their code. With such an RI schedule, it is possible for teams to spend the best part of a day to resolve their P0 and potentially their P1 issues resulting from failed RITs. If DDRITS-related bugs cannot be fixed within that timeframe, main product builds will be reverted to previous day's state.

Reporting is an important cornerstone of monitoring adherence to the original selection criteria as we enter the new shipping cycle. It will also act as the trigger mechanism for updating the existing test suite, if necessary. The central engineering team will be providing reports that examine the health of the integration suite by trending on the following types of data:

- **DDRIT /PURIT passing rates** (identifying who is to be in "test result hell")
- **Code velocity cadence** (identifying outliers, which teams are slow and fast to RI/FI)
- **Product and test gaps** (types of failures, customer scenarios or integration points being missed by which test suites)

3.2.2 Updating the Integration Test Suite

Given the above workflow and trending data/reports, we are currently addressing the challenge of updating and evolving the new integration test suite, once the product churns. During such churn, when significant feature enhancements or new customer scenarios are added, feature code as well as their integration points would be affected. The test suite would need to be adapted. Two scenarios are possible:

- **Integration test(s) fail during RI** – in this situation, *previously traversed* product features and integration points have been significantly modified or deleted in the RI'ing product branch to the

extent that one or more tests fail. As a result, the PU test owner needs to significantly rewrite one or more tests. For the DDRITs, this would result in new reviews of the updated tests by the divisional committee who, upon approval, would request renewed purification of that test case. For the PURITs, two PU RI reps need to decide whether the tests need to re-reviewed and/or re-purified. Details should be specified in the team's service level agreement.

- **Integration tests pass, but breaking changes occur upon FI** – in this situation, teams taking fearless FIs experienced failures due to significant modifications or deletions to existing product features and integration points. These features or integration points were *not previously traversed* by integration tests and given the severity of breaking changes, it may be decided that new integration tests need to be implemented or old ones modified/enhanced.

4. Supporting Tools and Infrastructure Work

We realized early on that significant changes were needed to our existing test infrastructure as well as additional tool support to support the definition, implementation, execution and most importantly updating of this integration test suite. In this section, we wanted to highlight three key improvements that are currently under way.

4.1 New Static Analysis and Dynamic Testing Tools

We are developing a set of static analysis and code coverage-based tools that enables us to capture and visualize function-level dependencies between product branches. We can then overlay those views with code coverage and code churn data. This enabled us early on in the initiative to support teams with identifying dependencies during the test definition phase and will in future enable us to validate the coverage of key integration points as part of test execution. Code churn data will be used to address the update issue by highlighting public APIs that have changed and thus integration points that may have been affected. For example, by analyzing the dependencies between teams A and B, we may identify that no dependencies exist or have been modified during the RI of team A's code, so team B can take their FI with a high degree of confidence that they will not be broken by team A. Alternatively, the tools may indicate that team B does have dependencies on team A's RI'ing code, but coverage data indicates that no RIT coverage is available. Team B can plan for some additional testing, suggest a new PURIT test to team A, if the dependency is unique or propose a new DDRIT to the division, if that dependency is taken by most teams.

4.2 API-based Test Automation Framework

Having learned from past and often painful experiences, our existing UI-based test automation needed to be complemented with an effective and efficient API-based approach for developing these integration tests, in particular, the DDRITs. The challenge was also one of consolidation as over the years, a plethora of API-based automation frameworks had emerged and evolved. The resulting framework helped us standardize around one tool, provided a uniform access mechanism to the Visual Studio Object Model (OM) and implemented standardized, yet extensible test code logging mechanisms. With access to the OM, the resulting tests cases demonstrate significant performance gains over the UI-based tests making the DDRITs tests fast and effective.

4.3 Test Portability

In the past, our existing integration test suite suffered from test portability-related issues. Teams were not necessarily able to run each other's tests early on as part of the RI process and if they did, it required considerable resources to execute and debug those tests. However, one of the requirements for the new integration test suite was that tests should be portable meaning that they can be *run by anyone*. This requirement has led to RITs test owners clearly specifying as well as reviewing and rationalizing the execution needs, dependencies, etc. for each test. As a result, integration tests can be run in the cloud by our central engineering team or any staff member across the division without additional effort.

5. Conclusion and Future Work

In this paper, we described an ongoing test improvement initiative whose goal is to help reinvigorate our code flow, both in terms of velocity and quality. We examined the key challenges facing us with large-scale integration testing at Microsoft and described our response in terms of the development of a new integration test suite. Details were given concerning the definition, implementation and execution of this new test suite as well as an overview of the anticipated workflow that leverages the test suite. Moreover, we outlined the tools and infrastructure changes that were required to support this effort.

At the time of writing, we have started to execute our new integration test suite regularly against the existing product code base and will be validating the results reported against our stated deployment goals. Based on those results, we will be focusing on developing more supporting tools and infrastructure that can help us evolve the existing tests.

6. Acknowledgements

I would like to thank Larry Sullivan, Director of Engineering and David Sauntry, Principal Architect Lead, for their ongoing support. I also want to express my gratitude to Steve Arnold, Richard Kuhn and Mark Osborne for their support and discussions regarding the issues discussed in this paper. Thanks also go to my reviewers whose invaluable feedback and comments are greatly appreciated.