Microsoft Dynamics® AX 2012

# Testing best practices for Microsoft Dynamics AX 2012

White Paper

This paper outlines testing best practices that are provided by the Microsoft Dynamics AX 2012 internal development team. These practices apply both to the development of product extensions that are built by independent software vendors (ISVs), and to customizations that are performed by Microsoft Dynamics AX implementation partners and customers for specific ERP deployments.

Date: September 2011

www.microsoft.com/dynamics/ax

Dave Froslie, Test Architect

Send suggestions and comments about this document to adocs@microsoft.com. Please include the title with your feedback.

Microsoft Dynamics

# Contents

# Introduction

This paper outlines testing best practices that are provided by the Microsoft Dynamics® AX 2012 internal development team. These practices apply both to the development of product extensions that are built by independent software vendors (ISVs), and to customizations that are performed by Microsoft Dynamics AX implementation partners and customers for specific ERP deployments.

Ensuring that an ERP application works after it is deployed at a customer site is a challenge. The base application, which is both broad and deep, has a myriad of modules, features, and software and hardware configurations. ISVs extend or modify this base application to provide functionality for specific market needs and verticals. Finally, implementation partners or customers perform point customizations to address specific needs. This combination of base application, one or more ISV add-ons, and point customizations that operate in a unique hardware and software configuration is what businesses depend on for mission-critical financial data and company operations.

The Microsoft Sure Step Methodology provides basic testing guidance that can be scaled for five project types. The terminology and process steps in Sure Step are the foundation of this paper. The best practices that are presented here supplement the Sure Step guidance for testing efforts in the Microsoft Dynamics AX ecosystem.

ISV development efforts can also benefit from the practices that are presented in this paper, because the practices support the requirements in the Certified for Microsoft Dynamics (CfMD) Solution Test. For more information, see Microsoft Dynamics Testing for ISVs.

# Don’t test quality in

A common misconception is that software testing starts after the product is built. If no other lesson is taken from this paper, let it be that you cannot “test quality” into the product. A primary goal of testing is to provide feedback about the product as soon as possible. Identifying issues in the requirements phase prevents them from becoming part of the design. Identifying issues in the design phase prevents them from being coded. Identifying issues during implementation prevents them from becoming part of the shipped product or customer deployment. A table in Steven McConnell’s book *Code Complete, Second Edition*, shows the average cost of fixing defects, based on when they are introduced and detected. For example, a defect that is introduced in the architecture phase costs 10 times as much to fix if it is detected in the construction phase, 15 times as much if it is detected during the system test, and 25 to 100 times as much if it is detected post release.

One point to emphasize is that the role of software tester must be filled on a project team. However, it does not have to be filled by a specific person or a team of dedicated software test professionals. For example, a dedicated software test lead can provide guidance and project management services to part-time software testers across multiple projects. In even smaller organizations, all testers may have to fill multiple roles. Even in this situation, it is important to identify someone who is responsible for software testing and quality during all phases of the project.

## Managing the project

Good project management and configuration management practices have a large impact on the quality of the project output. Practices should be established for artifacts such as the following:

- Requirements
- Source code
- Test cases
- Test code
- Bugs
- Work items

Although practices can be established for these artifacts by using a combination of tools, such as Microsoft SharePoint and Microsoft Office applications, a tool such as the Visual Studio Application Lifecycle Management (ALM) system (Visual Studio Application Lifecycle Management) drives higher quality and productivity. All the artifacts in the previous list can be effectively managed by using Microsoft Visual Studio / Team Foundation Server (VS/TFS) 2010. One of the most powerful features of this system is the capability to link artifacts for traceability. For example, a typical development process for a feature might have the flow and interrelations that are shown in Figure 1. In this example, Simon is a functional consultant, and Isaac is an IT developer.

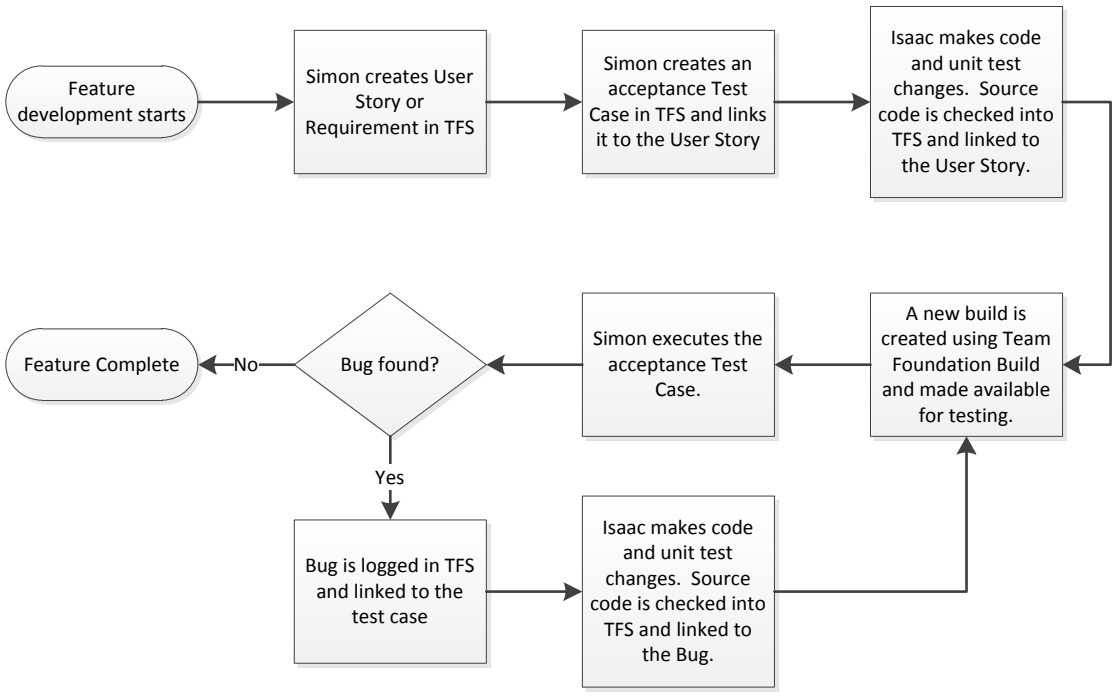


**Figure 1 Project management by using Team Foundation Server**

The process in Figure 1 involves much more than testing. It includes basic configuration management practices, such as source code control, bug tracking, and build processing. It integrates project management and requirements management into the development process. Both the traceability of these core artifacts and the integration between them are very powerful, and are an essential part of the quality practices in an organization.

For a description of the economic and other benefits of using the Visual Studio Application Lifecycle Management system, see the white paper The Business Case for Visual Studio Quality Assurance and Testing Tools.

## Peer reviews

A primary tool during the requirements, design, and coding phases is peer review. Software inspections, which are a rigorous approach to peer reviews, are defined as "peer review of any work product by trained individuals who look for defects using a well-defined process" (Wikipedia). Alternatively, peer reviews can be informal meetings to discuss the design of a particular new feature. The most important aspect of peer reviews is that multiple people think about, and work on, the same problem, and the focus is on identifying defects that can be prevented before the next phase of development.

Code review is perhaps the most important peer review that an organization can perform. As with other peer reviews, the formality can vary. The most formal type is code inspection. This typically involves a group that gets together in a meeting room to discuss the code line by line. Similar in the

level of detail, but much less formal, is pair programming. This involves two individuals who sit side by side, sharing one keyboard and monitor. Pair programming is essentially real-time review as two people think about the code simultaneously.

Informal code reviews that are facilitated through e-mail can be very effective. The developer of the code sends a small group of individuals an e-mail that contains the locations and details of the code changes. The group, which typically includes other developers and testers who are familiar with the area of code that was changed, review the changes that the developer made. The reviewers provide feedback in e-mail messages to the entire code review group. The authoring developer responds to all the feedback by e-mail. The developer's response for each issue falls into one of the following categories:

1. The issue has been fixed.
2. I do not want to fix the issue.
3. I would like to fix the issue, but this is not the appropriate time.

A work item is generated so that issues that fall into category 3 can be addressed in the future.

One positive side effect of any peer review approach is the shared learning that occurs between team members. Junior team members learn from experienced team members, and new ideas can be brought forward by anyone who participates. The overall knowledge of the team increases through these reviews, enabling better discussions over time.

## Static analysis

Static analysis tools evaluate the software code (source or object). Tools such as the Microsoft Dynamics AX Best Practice Checks (Best Practices for Microsoft Dynamics AX Development) and Visual Studio Code Analysis (Best Practices for Microsoft Dynamics AX Development) identify potential violations of programming and design guidelines.

Like peer reviews, static analysis tools catch quality issues early in the development. This prevents costly downstream discovery of issues when the software is tested. In addition to the predefined checks, checks that are created for specific issues can be performed. Static analysis tools can be configured so that they run before code check-in and reject a check-in if the code contains errors. For example, Figure 2 shows the **System settings** form for version control in Microsoft Dynamics AX 2012.

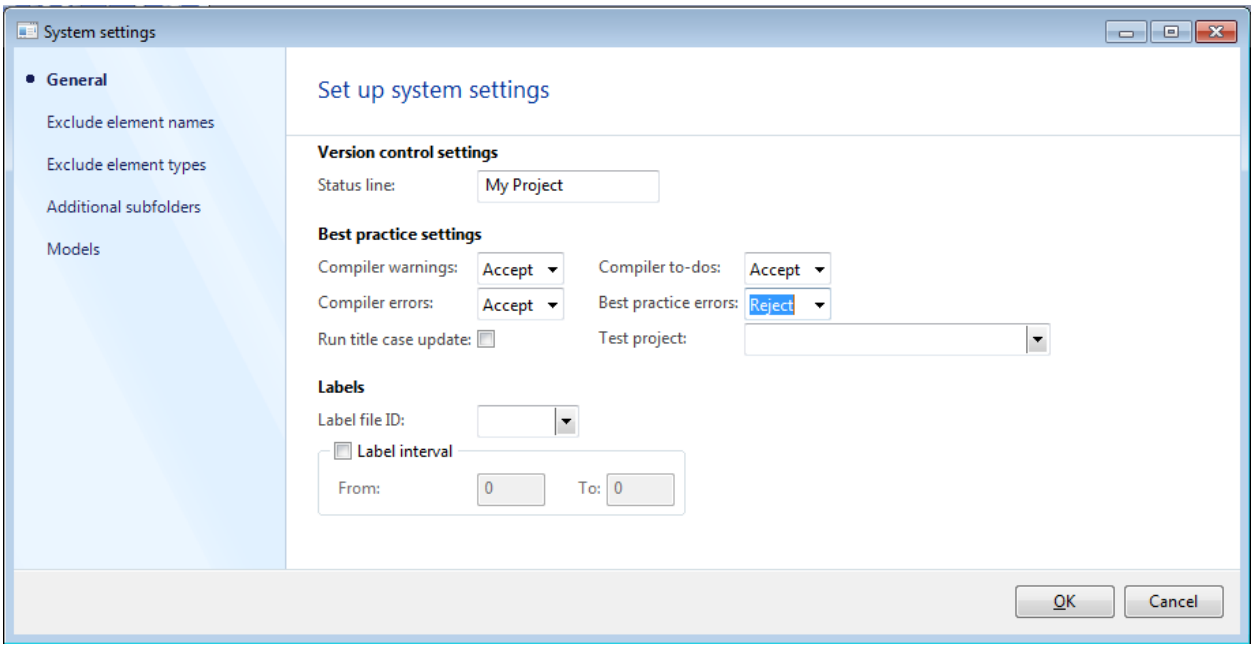


**Figure 2 System settings form for version control**

One enhancement that was made to the best practices in Microsoft Dynamics AX 2012 was the Form Style Analysis tool. This tool evaluates a form against the user experience guidelines for Microsoft Dynamics AX 2012. It can also fix the violation at the click of a button.

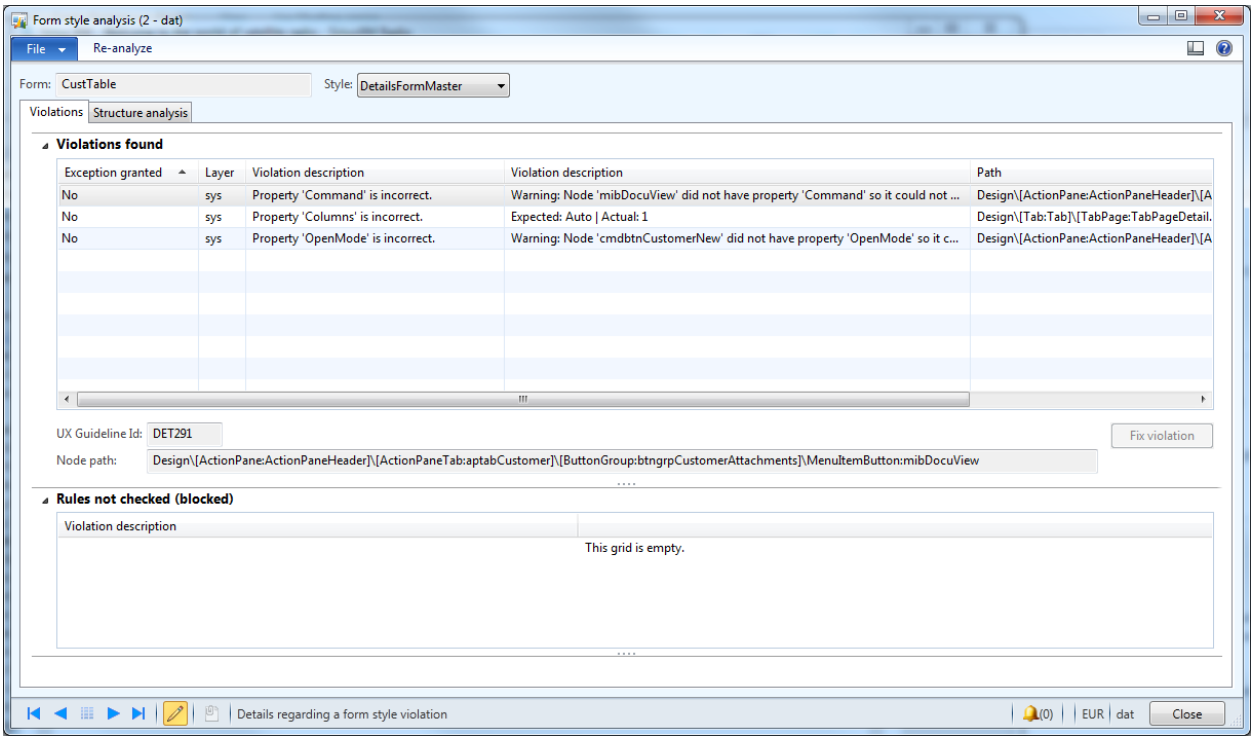


**Figure 3 Form Style Analysis tool**

Compliance with the best practices static analysis is a critical component of the Software Solution Test required for ISV product certification.

## Security

Proper design and implementation of security is a critical aspect of deploying an ERP system. The Microsoft Security Development Lifecycle (SDL) is a valuable tool for successful security implementation. The Microsoft Security Development Lifecycle web site has an excellent process description and tools that support proper security throughout the development life cycle.

Microsoft Dynamics AX 2012 introduces a new role-based security framework for securing the application and associated data. The framework enables a more cost-effective and a higher-quality approach to ensuring appropriate security. For more information, see What's New: Security.

# Testing phase best practices

Although focusing on the prevention or removal of defects early in the development cycle is a critical activity, it is clearly not enough. There are many ways for developers and testers to provide feedback about the application after coding begins.

As noted in the introduction to this paper, Microsoft Sure Step provides a thorough methodology and a set of templates to support the test phase. The schematic drawing in Figure 4 is taken directly from Sure Step section 1.6.1, "Gather Quality and Testing Standards" (Sure Step content version 3.4.9.0). The practices that are described in this section supplement Sure Step by providing more details and guidance for unit testing, function testing, subprocess testing, process testing, integration testing, and user acceptance testing.

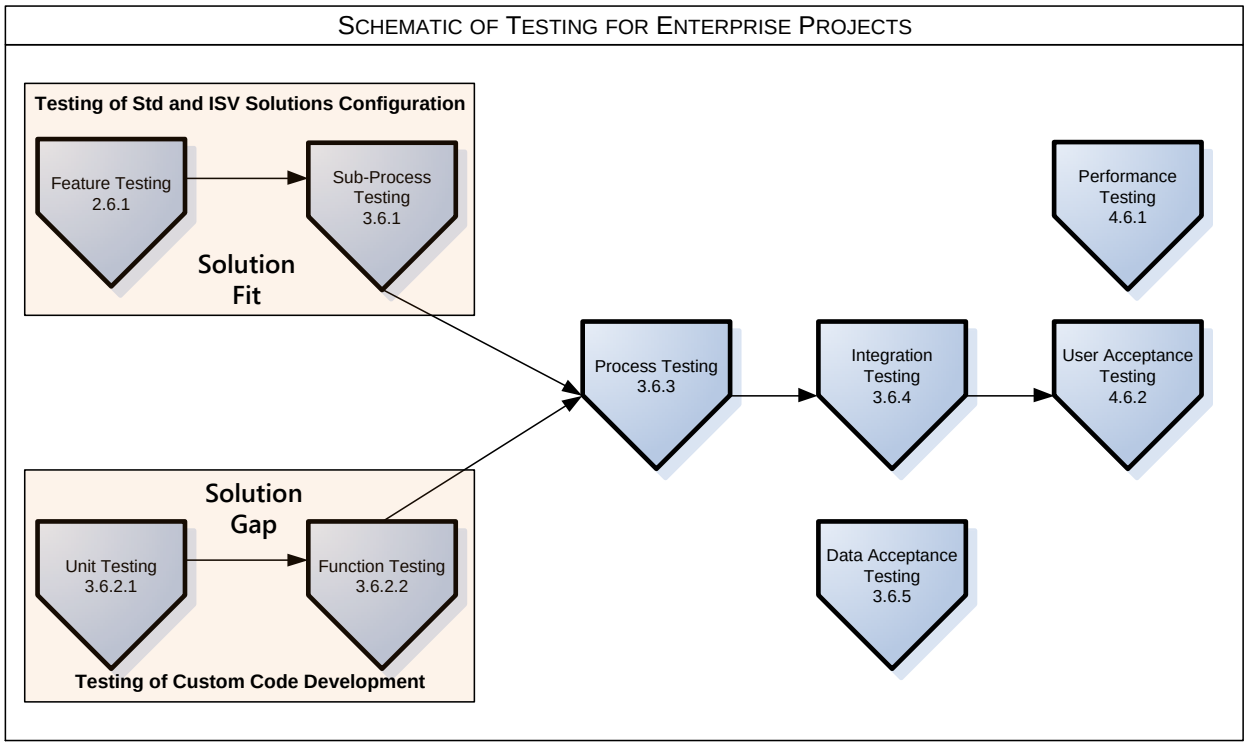


**Figure 4 Schematic from Sure Step section 1.6.1**

## Unit testing

Sure Step defines unit testing as "stand-alone testing of the system modification (custom code), performed during Development, by the Development Consultants." In software engineering, unit testing has grown significantly in the past few years. This is a bottom-up testing approach in which

automated tests are written by the developer. By contrast, in test-driven development, the test is written before the production code that provides the new or changed functionality.

**Note:** Although a unit test can be a manual test that is created and run by the developer, in software engineering, the term unit testing typically refers to automated tests.

Justifying the development of unit tests in parallel with product code can be a challenge. One argument against unit testing is that it takes highly paid developers away from writing the production code. Although this is a reasonable argument over the short term, the benefits of writing unit tests are significant. These benefits include finding bugs earlier, providing a safety net of tests for changes that are made later, and improving design. Over the long term, unit testing improves customer satisfaction and developer productivity.

A number of publicly available frameworks that enable unit testing have been written for various languages. For example, Visual Studio can implement unit tests for .NET languages. Because of the popularity of unit testing and the need for a framework that supports the X++ language that is used by Microsoft Dynamics AX, the SysTest unit testing framework was developed and shipped as part of Microsoft Dynamics AX 4.0. Key enhancements were made to this framework for Microsoft Dynamics AX 2012. For more information about SysTest, see Unit Test Framework.

The development of SysTest-based unit tests in Microsoft Dynamics AX is an important quality practice for ISVs and implementation partners. These unit tests, in combination with the X++ code coverage capability in the application, let developers and testers make informed decisions about the gaps in their testing process. Likewise, developing unit tests for managed code in the Visual Studio MSTest framework and leveraging the Visual Studio code coverage capability provide a powerful tool for managed code development.

## Function testing

Sure Step defines function testing as "stand-alone testing of the system modification (custom code), performed during Development by the Customer and the Consultants." As new functionality (also called a feature) is developed, testers should validate that the requirements are being met by that functionality. Testing at the feature level enables a fast turnaround on defects, which improves the efficiency of the development process.

As the Sure Step definition implies, the majority of the function testing of features in the Microsoft Dynamics AX ecosystem is done by domain experts. These domain experts have various titles – for example, functional consultant, business analyst, department head, functional solution architect, and power user. Note that none of these titles contain the word "tester." Testing is a part-time job for these individuals.

Because of their deep domain knowledge, these part-time testers are good at the hands-on definition and execution of test cases. However, they frequently need help developing and tracking test plans, managing test collateral, creating environments for testing, and tracking the overall software quality. Because of this, having an individual who acts as a test lead for the project is a wise investment. The test lead can bring order to what could turn into testing chaos if it is not properly organized.

Visual Studio 2010 brings some powerful testing tools to the Microsoft Dynamics AX ecosystem. The new Microsoft Test Manager (MTM) application is specifically targeted at a test team that is made up of domain experts. MTM contains functionality for the entire testing cycle, and is an enabler for both a test lead and the functional testers. A description of the capabilities of MTM is outside the scope of this paper. For more information about how to use MTM, see Testing the Application. MTM is available in the Visual Studio Ultimate and Visual Studio Test Professional packages.

Significant work went into Microsoft Dynamics AX 2012 to enable the data collectors in MTM to accurately record user actions when a test is run. The tester can then "fast forward" through the test case when it is rerun later.

## Subprocess, process, integration, and user acceptance testing

Subprocess, process, integration, and user acceptance testing are all forms of broader-scope tests, as defined in Sure Step. After the new functionality is validated in isolation, as described in the previous section, it has to be validated as part of a business process or user workflow. The scope of these tests varies. One test might be a single-user task that is performed independently, whereas another test might encompass several users or roles in the system as it tracks a workflow across the company's activities. Examples of the latter include the "quote to cash" and "procure to pay" processes.

The Microsoft Dynamics AX team used end-to-end (E2E) scenarios as part of its validation for the Microsoft Dynamics AX 2012 release. These scenarios are larger-scale integration tests that test a complete business cycle across multiple roles in the organization. In addition to functional correctness, these scenarios were evaluated for performance, usability, and completeness, and they were given an overall score based on all these inputs.

Each E2E scenario is described in three tiers:

- Tier 1 is a product-agnostic description of the business process.
- Tier 2 is a product-related persona-based flow of the business process.
- Tier 3 is a product-specific test script that contains a detailed walkthrough of the scenario in Microsoft Dynamics AX 2012.

The E2E scenarios that were used internally are being made available to the Microsoft Dynamics ecosystem, for use as templates and starting points for process, integration, and user acceptance testing.

The features of Microsoft Test Manager that were described in the previous section apply equally well to this section. Furthermore, the steps in the tier 3 test scripts of the E2E scenarios can be imported into TFS as test cases, and either run as one scenario or used in multiple scenarios.

## Test-driven development

Test-driven development practices optimize the feedback process during feature development. There are two complementary practices, both of which can apply to Microsoft Dynamics AX testing: developer-focused test-driven development (TDD) and acceptance test–driven development (ATDD). In Figure 4, the work that Simon and Isaac performed is typical of the development flow when TDD and ATDD are used. These practices fit into to the "Solution Gap – Testing of Custom Code Development" box in Figure 4.

TDD is the best-known test-driven practice and is a developer activity. When applying TDD, a developer first writes a failing unit test (a "red" state in the test tool), writes the required product code to make the test pass (a "green" state in the test tool), and then refactors the product and test code. This process is commonly referred to as the "Red-Green-Refactor" cycle. TDD drives good design and testability of the product code. In many ways, it is a design activity that has a thorough unit test suite as its byproduct. TDD can be accomplished by using the SysTest framework in Microsoft Dynamics AX.

When applying ATDD, a tester defines an acceptance test for the development effort before the coding starts. The up-front conversation drives clarity into the development process. Whether automated or not, the documented acceptance test provides an open-book test for the developer to work against. The acceptance test is run before the developer checks the code into the source code repository, to ensure that the feature is functional.

TDD provides an inner loop of feedback, in which the developer runs unit tests many times a day. ATDD provides an outer loop of feedback that validates customer requirements. The Visual Studio 2010 test toolset can be used together with Microsoft Dynamics AX for ATDD.

Although the two practices can be used independently, using both enables a test-focused and quality-driven culture for the development.

## Data management

Effective data management is critical for ERP testing efforts. Data sets must be sufficiently complex and large to enable effective functional validation, but not so large that deploying the data for test systems is excessively time consuming.

Throughout the Microsoft Dynamics AX 2012 development cycle, the development team created and maintained a data set that struck a balance between functional completeness and size. This data set, known as the Contoso data set, is being made available externally as demo data, together with instructions for loading the data. This data set is a good starting point for either an ISV product development effort or the early phases of a new implementation.

Most implementations, particularly upgrades, use a "scrubbed" copy of company data for development and testing. This can be very effective, but the size of the data can be a challenge. Limiting the transaction history is a good way to keep the size manageable.

One key to effective testing is ensuring that the system is in a known state when a test is started. This is especially true for automated testing, because a human who runs a test manually can more effectively deal with an unknown state than a computer, which requires a specific state. The following are some strategies for effectively maintaining a known state at the start of a test:

- Reset the system to a known state at the *start* of each test or group of tests. For simple tests that affect only part of the system, scripts can be written to clean specific tables and restore a base set of data. For more complex scenarios, an effective approach is to maintain a database backup and restore it. The Microsoft Dynamics AX development team creates "save points" of the database in the desired states, and then restores these save points in the setup portion of a test group.
- Reset the system to a known state at the *end* of each test or group of tests. For SysTest-based tests, built-in functionality enables SysTest to track changes that are made during the test, and then restore the system to the pretest state during test clean-up. For information about this functionality, see the MSDN documentation about the SysTest framework.

## To automate or not to automate?

Regression testing is "any type of software testing that seeks to uncover new errors, or regressions, in existing functionality after changes have been made to the software, such as functional enhancements, patches or configuration changes" (Wikipedia). The most common approach to regression testing is to rerun previously run tests to verify that the application's behavior has not changed. Ideally, there would be an automated test tool that software testers could use to run the same tests repeatedly.

Unfortunately, automation tools, particularly record and playback tools, have a mixed history in software engineering. The tools have historically been unstable, because of sensitivities in the execution environment and other factors. They often create volumes of code that is very difficult to maintain. Frequently, the maintenance costs quickly exceed the cost savings.

Companies that successfully create automated regression suites have a strong technical presence on their team. For example, the very title that is used for the "test engineer" at Microsoft, software development engineer in test, acknowledges that individuals who do testing must have very strong development skills. It is not uncommon for test engineers to end up writing more test code than the code that is contained in the product.

For teams that do not have a strong programming skill set on their test team, the capabilities of Microsoft Test Manager in Visual Studio 10 help improve the productivity of testers who do manual regression testing. By creating action recordings when running through a test case, the tester can fast forward to the interesting part of the test case when it is rerun later. If the application changes, or if there is some error during the playback, the tester can quickly revert to manually running or re-recording the test steps. Using shared steps enables central maintenance of steps that are common to many test cases.

For teams that have a strong programming skill set, the first task that can be automated is the validation of business logic, by using either the SysTest unit test framework or the MSTest unit test framework. Tests that target business logic below the user interface are faster, more reliable, and easier to maintain than tests that target the user interface.

Visual Studio 10 also provides the capability to create coded UI tests that target the user interface of an application. This capability can be used to create basic tests for Microsoft Dynamics AX, although this should be considered an advanced approach that may require non-trivial engineering work to be successful.

In summary, the recommendation for regression testing of Microsoft Dynamics AX is to use SysTest-based or MSTest-based tests to validate business logic, and to leverage the fast forward capability in Microsoft Test Manager for effective manual testing through the user interface.

# How we test software at Microsoft Dynamics

In 2009, Microsoft Press published a book titled *How We Test Software at Microsoft* (HWTSAM), written by three senior testers at Microsoft (Alan Page, Ken Johnston, and BJ Rollison). As the title suggests, the book provides background about the role of testing at Microsoft. It describes the processes, people, tools, and techniques that work together to ensure quality in Microsoft products.

Much of what is written in *How We Test Software at Microsoft* applies to how we test software in the Microsoft Dynamics AX team, but there are some specific challenges to testing a broad, business-critical, and highly customized application such as Microsoft Dynamics AX 2012. We are often asked how we ensure quality in our development process, and this part of the paper provides some insight into this issue.

The practices that were described in the first part of this paper are practices that the Microsoft Dynamics AX team followed during the Microsoft Dynamics AX 2012 development cycle, although some of the specifics and tools are different, because of the scale and make-up of the team. This part of the paper highlights some of the differences and describes some additional best practices that the team uses. Not all these practices apply to smaller-scale development, but they are included for informational purposes, and because they may inspire ISVs or implementation partners to apply a similar approach to their development efforts.

For consistency, the primary sections from the first part of the paper are used to frame this section, although subsections are added or removed, depending on the best practices that are highlighted.

## You cannot test quality into a product

As described in HWTSAM, Microsoft divisions are typically organized into feature teams that consist of members from the three primary development disciplines at Microsoft: program managers (PMs), software development engineers (SDEs), and software development engineers in test (SDETs). A feature team has responsibility for an area of the system. For example, in Microsoft Dynamics AX, there are feature teams for Accounts payable, Server, Expense management, and many more areas. Although each engineering discipline is responsible for unique deliverables in the engineering process, the feature team collectively owns the quality of the feature. Each discipline brings its perspective to every phase of development, from requirements through testing. These unique perspectives are critical to ensuring a quality focus from the start of feature development.

### Managing the project

During Microsoft Dynamics AX 2012 development, the core planning unit for the Microsoft Dynamics AX feature team was the feature. The set of features was derived from high-level scenarios or themes, and went through several reviews before the final set of committed features for a major milestone was determined.

Before implementation began, a feature was broken down into smaller units, called testable units (TUs). A TU typically represents less than one week of development work for an SDE. The TUs were estimated, and a development plan was created for the feature. For each TU, the SDET created a

scorecard test to verify basic functionality. Although the SDE and the SDET owned the specific deliverables, this effort was done collaboratively across all three disciplines.

The features and TU work items were created in a Team Foundation Server system that was created for the managing the project. These work items were the major progress tracking tool during the development phase.

An important part of completing a milestone was the creation and review of milestone exit criteria. These criteria formed the "definition of done" for the milestone, and were made up of product criteria and engineering criteria. The following are some examples of the exit criteria:

- Product criteria:
  - All functional test cases (automated and manual) are run, failures are reviewed, and bugs are created for all failures.
  - All upgrade test cases (automated and manual) are run, failures are reviewed, and bugs are created for all failures.
  - Targeted functional test cases are run in selected international environments, failures are reviewed, and bugs are created for all failures.
  - All bugs that meet a targeted severity and priority must be fixed and retested.
  - Performance targets are met.
  - Accessibility test cases are run, failures are reviewed, and bugs are created for all failures.
  - Software Design Lifecycle (SDL) requirements are met.
  - End-to-end scenarios are run and meet targeted quality goals.
  - Geopolitical scans are run, and no issues occur.
  - And many more.
- Engineering criteria:
  - No static analysis errors or warnings occur.
  - The percentage of priority 1 test cases that are automated must meet a target.
  - The percentage of all test cases that are automated must meet a target.
  - Code coverage for each area of the system should meet a target, and coverage gaps should be reviewed.
  - And many more.

### Peer reviews

Several documentation deliverables were created and reviewed by a cross-discipline team before the coding effort started:

- A Functional Specification that contains scenarios, business requirements, a conceptual object model, a logical data model, functional requirements, use cases, and sections about several non-functional areas, such as security, performance, and extensibility.
- A Development Design that contains an architecture and design overview, class and interface descriptions, a detailed design for various Microsoft Dynamics AX types, unit testing plans, and other design issues, such as setup and globalization.
- A Test Design Specification that contains the test strategy, automation plan, manual test plan, test data requirements, and detailed plans for functional testing, integration testing, security testing, upgrade testing, and other testing types.

Code reviews were required for all product check-ins to the source code control system, and for most test check-ins.

### Static analysis

Static analysis is a critical part of the internal development process. As part of the source code check-in process, the SDE was required to remove all Best Practice Check errors and FxCop errors. No compiler warnings were allowed.

### User experience

A dedicated User Experience team engages with customers and the development team throughout the release. Testing of different user interface prototypes in User Experience Labs ensures optimal usability in the released product.

After a user interface pattern is finalized, implementation across the product must adhere to a set of best practice checks in the Form Style Analyzer, as described earlier in this paper.

### Technology Adopter Program

The Technology Adopter Program (TAP) was used to enable customer feedback throughout the development cycle. The TAP consisted of key customers, implementation partners, and ISVs. This group met regularly throughout the development cycle to review and discuss specifications, designs, and plans with the development team. The group also received regular code drops of the in-progress release and provided feedback.

## Test phase best practices

### Unit testing

SDEs created new unit tests and modified existing unit tests while developing new features. The source code check-in process required a minimum level for the code coverage of the unit tests that the developer created. For X++ development, these tests were developed by using the SysTest framework in MorphX. For managed code, these tests were developed by using an internal test harness and/or MSTest. Tens of thousands of unit tests were run during Microsoft Dynamics AX 2012 development. The number of lines of code for unit testing was nearly the same as the number of lines of code for the product.

A subset of the unit tests was defined as check-in tests (CITs). These tests were run and required to pass as part of the gated check-in system that all SDEs used for any check-in to the source code repository. This prevented major regressions from being introduced into the code base.

### Function testing

The first hands-on use of a feature by the SDETs came as part of the scorecard testing of the testable units. This testing is a lightweight form of acceptance test–driven development (ATDD). SDETs and SDEs work very closely in this phase of the project.

### Subprocess, process, integration, and user acceptance testing

In addition to testing that was focused on features, the SDETs broadened their testing efforts to focus on key interfaces with other areas of the system. As described earlier, end-to-end scenarios were a key part of this testing effort.

In addition to the scripted E2E scenarios, the test team coordinated numerous "interactive test sessions" in the later phases of the release. The goal of each session was to bring together SDETs, PMs, and SDEs to focus on a particular business cycle or a particular area of the product. These sessions were critical to driving completeness into the product.

### To automate or not to automate?

Because of the scale of the product and the long-term support requirements for Microsoft Dynamics AX, the development team made a heavy investment in automated regression tests during the Microsoft Dynamics AX 2012 development cycle. The unit tests were one part of this regression suite. Additionally, many functional tests were automated. For these tests, the user interface of the product

was the primary interaction point. The automated regression tests for features fell into one of three categories:

- Build verification tests (BVTs) – These tests verify the most basic functionality in the product and can be considered "smoke tests." These tests were run as part of the gated check-in system for every SDE change. Tens of BVT test cases were executed thousands of times during the development cycle.
- Build acceptance tests (BATs) – These tests verify core functionality in each functional area of the product. As core features of the product were created, a new test case was created, or an existing test case was modified. These tests were run together with every nightly build. They also were frequently run before an SDE check-in to verify that no functional breaks resulted from the check-in. Hundreds of BAT test cases were executed for each run.
- Weekly tests – These tests made up the remainder of the automated test suite. As the name suggests, these tests were run one time per week for much of the release. As Microsoft Dynamics AX 2012 approached its release to customers, the tests were run much more frequently. Tens of thousands of weekly test cases were executed for each run.

The team has several milestone and release goals for automation – for example:

- A targeted percentage of priority 1 test cases must be automated.
- A targeted percentage of all test cases must be automated.
- A targeted percentage of code coverage must be met for each area of the system.

Another category of regression testing is the verification of bug fixes. The workflow that is associated with a bug can be summarized as follows:

1. The bug can be created by anyone on the team and is mapped to a feature team in the bug tracking tool. The bug is in an **Active** state.
2. The bug is triaged by a cross-discipline team (PM, SDE, and SDET). The triage result is one of the following:
    - **Fix**
    - **Don't Fix** – The bug is put into a **Resolved** state, and a resolution must be specified. The resolution options include **By Design**, **Duplicate**, **External**, **Fixed**, **Not Repro**, **Postponed**, and **Won't Fix**.
    - **Vnext Candidate** – The bug is put into a **Resolved** state, and a resolution must be specified. The options are the same as the options for **Don't Fix**.
3. If the triage result is **Fix**, it is assigned to an SDE, who makes the changes that are required to address the issue. Upon check-in to source code control, the bug is in a **Resolved** state, and the resolution is **Fixed**.
4. Regardless of the triage result, all bugs that are in the **Resolved** state are assigned to an SDET so that the resolution can be reviewed. If the resolution is **Fixed**, the SDET tests the fix to verify correctness. The SDET also acts as a customer advocate, and provides a "check and balance" in the system for other resolution types. For example, an SDET may "push back" on a bug that has a resolution of **Postponed**, by providing more details or a stronger argument for the triage team to consider.
5. If the SDET supports the resolution, the SDET puts the bug into a **Closed** state. The SDET gets input from the original bug author before closing the bug. As part of the closing process, the SDET reviews the existing test case collateral and decides whether test cases must be updated to ensure that the product does not regression in the future.
6. Any bugs that have a resolution of **Postponed** are reactivated in the next release.

# Conclusion

Testing ERP applications and customizations of those applications presents a difficult problem. Although each ISV, implementation partner, and customer situation is unique, the best practices in this paper, in combination with the Sure Step Methodology, provide a basic framework for the development of a test strategy in the Microsoft Dynamics AX 2012 ecosystem. Following these practices ensures a more successful certification process for ISV products.

The capabilities of Visual Studio 2010 Application Lifecycle Management enable a number of best practices that prevent defects, and that support a well-managed, transparent project.

Visual Studio 2010 testing capabilities enable the full test cycle, from planning through test execution.

For automated regression testing, focus first on using the SysTest and MSTest test harnesses to verify business logic, and the Microsoft Test Manager fast forward capability to improve manual test execution of UI scenarios.

# Appendix

## Software testing resources

The software testing discipline is evolving, and resources for software testers are becoming more prevalent. Although this paper provides best practices for testing Microsoft Dynamics AX 2012, the following list provides a starting point for some good websites and books:

- Visual Studio 10 Information – Numerous resources are available for Visual Studio 10. The following are a few key links that are relevant to Application Lifecycle Management and testing:
  - Visual Studio Test Professional 2010 Tour – At this site, you can take a tour of the capabilities of Visual Studio Test Professional.
  - Testing Quality Tools Business Case White Paper – This white paper quantifies the economic and business benefits of applying the Visual Studio 10 quality and testing solution. It also discusses the benefits of agile development approaches.
  - Visual Studio Application Lifecycle Management Strategy – Learn about the strategic approach that Visual Studio is taking for current and future life cycle management products.
  - Visual Studio Application Lifecycle Management Capabilities – Learn about the core parts of the Visual Studio solution, including Test Management.
  - Professional Application Lifecycle Management with Visual Studio 2010, by Mickey Gousset, Brian Keller, Ajoy Krishnamoorthy, and Martin Woodward – From the book description: "*Written by Microsoft insiders, this nuts-and-bolts guide walks you through the tools, guidelines, and methodologies you'll need for Application Lifecycle Management (ALM) with Visual Studio 2010.*"
- Lessons Learned in Software Testing, by Cem Kaner, James Bach, and Bret Pettichord – From the book description: "*Decades of software testing experience condensed into the most important lessons learned.*"
- Testing Computer Software, 2nd Edition, by Cem Kaner, Jack Falk, and Hung O. Nguyen – From the book description: "*This book will teach you how to test computer software under real-world conditions*."
- A Practitioner's Guide to Software Test Design, by Lee Copeland – From the book description: *"Here's a comprehensive, up-to-date and practical introduction to software test design. This invaluable book presents all the important test design techniques in a single place and in a consistent, and easy-to-digest format."*
- xUnit Test Patterns: Refactoring Test Code, by Gerard Meszaros – From the book description: *"Automated testing is a cornerstone of agile development. An effective testing strategy will deliver new functionality more aggressively, accelerate user feedback, and improve quality. However, for many developers, creating effective automated tests is a unique and unfamiliar challenge."*
- Code Complete: A Practical Handbook of Software Construction, by Steve McConnell – From the book description: "*Take a strategic approach to software construction and produce superior products with this fully updated edition of Steve McConnell's critically praised and award-winning guide to software development best practices."*
- Dynamics AX Community Site – This site contains syndicated and hosted blogs, videos, news, articles, and an "Ask the Community" feature.
- Unit Test Framework at the Microsoft Dynamics AX Developer Center – Provides an overview of SysTest and links to "how to" examples.

## FAQ

1. Does Microsoft intend to provide a Record and Playback UI automation tool for Microsoft Dynamics AX 2012?

   ***Answer:*** As discussed in this paper, the Visual Studio 10 testing tools can be used to record actions that are taken while a test script runs. These actions can then be used to fast forward through the test for regression purposes.

2. The Task Recorder feature in Microsoft Dynamics AX 2012 provides the capability to record user steps and to export the steps, together with screen shots, to Microsoft Office documents. Can this be used for testing?

   ***Answer:*** Although the Task Recorder can be used to record test scripts, it is best suited to developing training material. The Microsoft Test Manager in Visual Studio 10 has a much fuller feature set for managing a test program, including the creation, maintenance, and execution of test scripts.

3. What Visual Studio packages are the testing tools, such as Microsoft Test Manager, available in?

   ***Answer:*** Visual Studio Ultimate and Visual Studio Test Professional.

4. Are there commercially available tools from companies other than Microsoft that can be used to automate Microsoft Dynamics AX 2012?

   ***Answer:*** The accessibility improvements in Microsoft Dynamics AX 2012 may enable interoperability with other commercially available testing tools, but no testing has been done for any of these tools.

5. What is the best way to get started with the SysTest unit test framework?

   ***Answer:*** There are several ways to get started with SysTest, including the MSDN site, Unit Test Framework, and the *Inside Microsoft Dynamics AX 2009* book, which has information about how to perform unit testing by using SysTest.

6. Where is the best place to go for more detailed information about the practices that are described in this paper?

   ***Answer:*** Additional information is provided through documentation and blog posts on MSDN and the Microsoft Dynamics AX Community Site. For blogs, expect more information to be published at http://blogs.msdn.com/b/dave_froslie/. The Community Site is the place to get updates from others in the ecosystem, and to ask and answer questions.

7. Does Microsoft intend to provide any of its internal tests so that they can be used for Microsoft Dynamics AX 2012 development that is done by ISVs, partners, or customers?

   ***Answer:*** As noted in this paper, the end-to-end scenarios that the team used during the development cycle are being made available in a Microsoft Excel format that can be imported into Visual Studio Team Foundation Server. There is no plan to provide any additional internal tests.

# Bibliography

McConnell, Steven. 2004. *Code Complete, Second Edition*. Redmond, WA. Microsoft Press.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

© 2011 Microsoft Corporation. All rights reserved.